aws

Developer Guide

Amazon Cognito

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon Cognito: Developer Guide

Copyright © 2026 Amazon Web Services, Inc. and/or its affiliates. All rights reserved.

Amazon's trademarks and trade dress may not be used in connection with any product or service that is not Amazon's, in any manner that is likely to cause confusion among customers, or in any manner that disparages or discredits Amazon. All other trademarks not owned by Amazon are the property of their respective owners, who may or may not be affiliated with, connected to, or sponsored by Amazon.

Services or capabilities described in AWS documentation might vary by Region. To see the differences applicable to the AWS European Sovereign Cloud Region, see the AWS European Sovereign Cloud User Guide.

Table of Contents

What is Amazon Cognito? 1
Which parts of Amazon Cognito do you need? 1
What you can build with Amazon Cognito 2
User pools 3
Identity pools 4
Features of Amazon Cognito 5
User pools 5
Identity pools 8
Amazon Cognito user pools and identity pools comparison 10
Getting started with Amazon Cognito 14
Regional availability 14
Pricing for Amazon Cognito 14
Terms and concepts 14
General 15
User pools 18
Identity pools 22
Getting started with AWS 23
Sign up for an AWS account 23
Getting started 24
Your first sign-in 25
Your first app and user pool 26
Other application options 28
React SPA example 29
Flutter mobile app example 32
Next steps 35
Add a social provider 36
Add a SAML IdP 43
Additional getting started options 46
Planning your design 48
Choose an approach 48
Common Amazon Cognito scenarios 50
Authenticate with a user pool 51
Access server-side resources 51
Access resources with API Gateway and Lambda 52

Access AWS services with a user pool and an identity pool ... 53
Authenticate with a third party and access AWS services with an identity pool ... 54
Access AWS AppSync resources with Amazon Cognito ... 54
Machine-to-machine (M2M) ... 55
Step-up authentication ... 55
Setting up your sign-in surface ... 57
Managed login ... 57
Managed login localization ... 59
Terms documents ... 60
Setting up managed login with the Amazon Cognito console ... 62
Setting up managed login with AWS Amplify ... 63
Viewing your sign-in page ... 63
Customizing your authentication pages ... 65
Things to know about managed login and the hosted UI ... 65
Configuring a domain ... 68
Branding and customization ... 83
Migrate to managed login ... 96
Authenticating users ... 98
Features ... 98
Sign-up ... 98
Sign-in ... 100
Managed login ... 101
Threat protection ... 102
Custom user experience ... 102
Monitoring and analytics ... 102
Amazon Cognito identity pools integration ... 102
User pool feature plans ... 103
Select a feature plan ... 105
Features by plan ... 106
Essentials plan features ... 109
Plus plan features ... 113
Turn off ineligible features ... 116
Authentication ... 117
App clients ... 118
Implement authentication flows ... 128
Things to know ... 131

Authentication flow example ... 133
Managed login authentication ... 136
SDK authentication ... 139
Authentication flows ... 145
SDK authorization models ... 171
Third-party IdP sign-in ... 186
Accessing resources ... 258
Accessing resources with Verified Permissions ... 259
Accessing API Gateway resources ... 261
Accessing AWS resources using an identity pool ... 262
User pool tokens ... 268
ID tokens ... 269
ID Token Header ... 270
ID token default payload ... 271
ID Token Signature ... 278
Access tokens ... 278
Access token header ... 279
Access token default payload ... 280
Machine-to-machine access token payload ... 282
Access token signature ... 283
Parse tokens safely ... 283
Refresh tokens ... 284
Refreshing tokens ... 285
Refresh token rotation ... 285
API and SDK token refresh ... 287
OAuth token refresh ... 289
Revoking refresh tokens ... 289
Revoking tokens ... 290
Enable token revocation ... 291
Revoke a token ... 292
Verifying a JSON Web Token ... 293
Prerequisites ... 293
Validate tokens with aws-jwt-verify ... 294
Understanding and inspecting tokens ... 295
Caching tokens ... 300
Caching machine-to-machine access tokens with Amazon API Gateway ... 301

M2M and scopes ... 304
API authorization ... 305
Machine-to-machine (M2M) authorization ... 305
About scopes ... 308
About resource servers ... 310
Resource binding ... 314
Managing users ... 316
Managing users ... 316
Allowing user sign-up ... 317
Signing up and confirming user accounts ... 320
Creating users as administrator ... 347
Adding groups to a user pool ... 354
Managing and searching for users ... 357
Passwords ... 365
Importing users into a user pool ... 371
Attributes ... 397
Updating a user pool and app client ... 415
Settings you can't change ... 416
SMS configuration ... 417
Updating a user pool with an AWS SDK, AWS CDK, or REST API ... 417
Customizing user pools ... 420
Lambda triggers ... 420
Things to know ... 423
Set up triggers ... 425
User pool Lambda trigger event ... 426
User pool Lambda trigger common parameters ... 427
Client metadata ... 428
Trigger sources by operation ... 431
Trigger sources by function ... 438
Pre sign-up ... 443
Post confirmation ... 450
Pre authentication ... 453
Post authentication ... 457
Inbound federation ... 461
Custom challenge ... 471
Pre token generation ... 493

Migrate user ... 516
Custom message ... 522
Custom senders ... 530
Messaging and analytics ... 546
Email settings ... 546
SMS message settings ... 560
Using Amazon Pinpoint analytics ... 576
Protecting user pools ... 582
Security best practices ... 583
Protect your user pool at the network level ... 583
Protect against SMS message abuse ... 584
Understand public authentication ... 584
Protect confidential clients with client secrets ... 588
Protect other secrets ... 588
User pool administration least privilege ... 589
Secure and verify tokens ... 592
Determine the identity providers that you want to trust ... 592
Understand the effect of scopes on access to user profiles ... 592
Sanitize inputs for user attributes ... 593
Adding MFA ... 593
Things to know ... 594
User MFA preferences ... 598
MFA logic ... 601
Configure MFA ... 606
SMS and email MFA ... 607
TOTP software token MFA ... 609
Authentication levels with ACR and AMR claims ... 615
Authentication levels and methods ... 616
Configuring ACR level names ... 619
Feature plan requirements ... 620
Your responsibilities for step-up authentication ... 621
Working with devices ... 621
Setting up remembered devices ... 622
Getting a device key ... 623
Signing in with a device ... 624
Viewing, updating and forgetting devices ... 626

Threat protection ... 627
Considerations and limitations for threat protection ... 629
Turning on threat protection in user pools ... 630
Threat protection enforcement concepts ... 631
Threat protection options ... 632
Prerequisites ... 633
Configuring threat protection ... 633
Compromised credentials ... 640
Adaptive authentication ... 641
Data collection ... 652
AWS WAF Web ACLs ... 656
Things to know about AWS WAF web ACLs and Amazon Cognito ... 656
Content-Security-Policy on custom block responses ... 658
Status codes in custom block responses ... 659
Associating a web ACL with your user pool ... 659
Testing and logging AWS WAF web ACLs ... 661
Case sensitivity ... 662
Deletion protection ... 664
Managing user disclosure ... 665
Authentication and user creation operations ... 666
Password reset operations ... 671
Confirmation operations ... 671
Multi-tenancy best practices ... 672
Per-tenant user pools ... 674
Per-tenant app clients ... 676
Per-tenant user pool groups ... 678
Per-tenant custom attributes ... 680
Per-tenant custom scopes ... 682
Example resource ... 685
Multi-tenancy security recommendations ... 686
Integrating with apps ... 687
Authentication with AWS Amplify ... 688
Creating a user interface (UI) with Amplify ... 689
Authentication with AWS SDKs ... 690
How authentication works ... 691
Managed login authentication ... 692

SDK authentication ... 694
Third-party identity provider authentication ... 698
Identity pool authentication ... 701
Working with AWS SDKs ... 704
Authorization with Amazon Verified Permissions ... 704
API authorization with Verified Permissions ... 706
Example policy for an Amazon Cognito user ... 709
User pool endpoints reference ... 711
Managed login endpoints ... 712
Login endpoint ... 715
Logout endpoint ... 717
Federation endpoints ... 720
OIDC issuer ... 722
Authorize endpoint ... 724
Token endpoint ... 737
userInfo endpoint ... 749
Revoke endpoint ... 752
SAML idpresponse endpoint ... 754
OAuth 2.0 grants ... 757
Using PKCE ... 758
How Amazon Cognito uses PKCE ... 758
Managed login and federation error responses ... 761
Deploying and operating ... 763
Quotas ... 763
Understanding API request rate quotas ... 764
Managing API request rate quotas ... 767
User pools request rate quotas ... 775
Identity pools request rate quotas ... 788
Quotas on resource number and size ... 790
Multi-Region replication ... 797
Things to know about multi-Region replication ... 797
Limitations of multi-Region replication ... 798
Configuring multi-Region replication ... 799
Supported API operations in secondary Regions ... 801
Failover in multi-Region user pools ... 803
Tagging resources ... 805

Supported resources ... 806
Tag restrictions ... 806
Managing tags with the console ... 806
AWS CLI examples ... 807
API actions ... 810
CloudFormation tagging ... 811
Troubleshooting ... 811
Custom domain configuration errors ... 812
`Invalid refresh token` error ... 814
Invalid SAML response errors in federation ... 815
Managed login users can't select an MFA factor ... 817
Not able to receive password reset code through email/SMS ... 818
Password reset fails with unverified recovery attributes: `Could not reset password for the account, please contact support or try again` ... 819
`SECRET_HASH` errors ... 820
The Amazon Cognito console chooses a default configuration for a new user pool ... 821
Additional troubleshooting resources ... 822
Code examples ... 823
Amazon Cognito Identity ... 825
Basics ... 826
Scenarios ... 850
Amazon Cognito Identity Provider ... 852
Basics ... 854
Scenarios ... 1021
Amazon Cognito Sync ... 1208
Basics ... 1209
Exchanging identities for AWS credentials ... 1211
Amazon Cognito identity pools ... 1211
Configuring identity pools ... 1213
Identity pools authentication flow ... 1220
IAM roles ... 1231
Security best practices ... 1249
Using attributes for access control ... 1254
Using role-based access control ... 1262
Getting credentials ... 1269
Using credentials ... 1275

Third-party identity providers 1277
Developer-authenticated identities 1308
Switching identities 1325
Getting started with identity pools 1328
Create an identity pool in Amazon Cognito 1329
Set up an SDK 1331
Integrate the identity providers 1331
Get credentials 1331
Example application 1332
Security 1366
Data protection 1367
Data encryption 1367
Encryption in transit 1377
Identity and access management 1378
Audience 1378
Authenticating with identities 1378
Managing access using policies 1380
How Amazon Cognito works with IAM 1381
Identity-based policy examples 1389
Troubleshooting 1394
Using service-linked roles 1396
Logging and monitoring 1401
Monitoring costs 1401
Exporting user pool logs 1404
Monitoring quotas and usage 1415
CloudTrail logs 1434
AWS PrivateLink 1462
Authentication flows for AWS PrivateLink integration 1463
Operational modes for AWS PrivateLink 1464
Considerations 1465
Controlling access with resource control policies 1470
Create an interface endpoint 1471
Create an endpoint policy 1471
Create an identity-based policy 1473
Compliance validation 1475
Resilience 1475

Regional data considerations 1476
Infrastructure security 1477
Configuration and vulnerability analysis 1477
AWS managed policies 1477
Policy updates 1479
Legacy 1482
Amazon Cognito Sync 1482
Amazon Cognito Sync availability change 1483
Getting started with Amazon Cognito Sync 1484
Synchronizing data across clients 1485
Handling event callbacks 1495
Implementing push synchronization 1512
Implementing Amazon Cognito Sync streams 1522
Customizing workflows with Amazon Cognito Events 1524
Using Amazon Cognito Sync with identity pools 1529
Managing datasets 1529
Bulk publish data 1530
Activate push synchronization 1531
Set up Amazon Cognito Streams 1531
Set up Amazon Cognito Events 1532
Hosted UI (classic) branding 1532
Specifying a custom logo in classic branding 1533
Specifying CSS customizations in classic branding 1533
Customizing the hosted UI with classic branding in the AWS Management Console 1537
Customizing the hosted UI with classic branding in the user pools API and with the AWS CLI 1538
Document history 1540

What is Amazon Cognito?

Amazon Cognito is an identity platform for web and mobile apps. It's a user directory, an authentication server, and an authorization service for OAuth 2.0 access tokens and AWS credentials. With Amazon Cognito, you can authenticate and authorize users from the built-in user directory, from your enterprise directory, and from consumer identity providers like Google and Facebook.

Topics

- Which parts of Amazon Cognito do you need?
- What you can build with Amazon Cognito
- User pools
- Identity pools
- Features of Amazon Cognito
- Amazon Cognito user pools and identity pools comparison
- Getting started with Amazon Cognito
- Regional availability
- Pricing for Amazon Cognito
- Common Amazon Cognito terms and concepts
- Getting started with AWS

The two components that follow make up Amazon Cognito. They operate independently or in tandem, based on your access needs for your users.

Which parts of Amazon Cognito do you need?

Amazon Cognito answers two different questions. Choose based on what your application must do.

Who is this user? (authentication)

> Use a user pool. A user pool is a user directory that draws from the OpenID Connect (OIDC) standard to authenticate users and issue JSON web tokens (JWTs) to your app or API. Users can sign in directly, through social providers like Google and Apple, or through enterprise SAML and OIDC providers. Most applications that need sign-up and sign-in start here.

What AWS resources may this user reach? (authorization)

Use an identity pool. An identity pool exchanges a trusted token (from a user pool, a social provider, or a SAML or OIDC provider) for temporary, limited-privilege AWS credentials so that your app can call services like Amazon S3 and Amazon DynamoDB on the user's behalf. An identity pool can also issue limited-privilege credentials to guest users who haven't signed in, without any token.

Do you need both?

Use them together when your app signs users in *and* needs to grant them direct AWS resource access: authenticate with a user pool, then exchange the user pool token with an identity pool for AWS credentials.

Neither component requires the other. A user pool can issue tokens to your own back end with no identity pool. An identity pool can broker credentials for a provider that you already have, even without a user pool. For more information about the differences between the two components, see Amazon Cognito user pools and identity pools comparison. To start building, see Getting started with Amazon Cognito.

What you can build with Amazon Cognito

Most Amazon Cognito applications fall into a few common shapes. Use this list to find the one closest to what you're building, then follow the link for the full walkthrough.

Sign users in to your app

A user pool authenticates your users and issues JSON web tokens (JWTs) that your app or API verifies. This is the starting point for most apps that need sign-up and sign-in. See Authenticate with a user pool.

Protect your own API or back-end resources

Use the access token from a user pool to authorize requests to your server-side resources, or let Amazon API Gateway validate the token and gate access to a Lambda function or your own API. See Access back-end resources with user pool tokens and Access resources with API Gateway and Lambda with a user pool.

Give users temporary access to AWS services

Exchange a user pool token with an identity pool for temporary, limited-privilege AWS credentials so your app can call services like Amazon S3 and Amazon DynamoDB on the user's behalf. See Access AWS services with a user pool and an identity pool.

Let users sign in with Google, Apple, or an enterprise directory

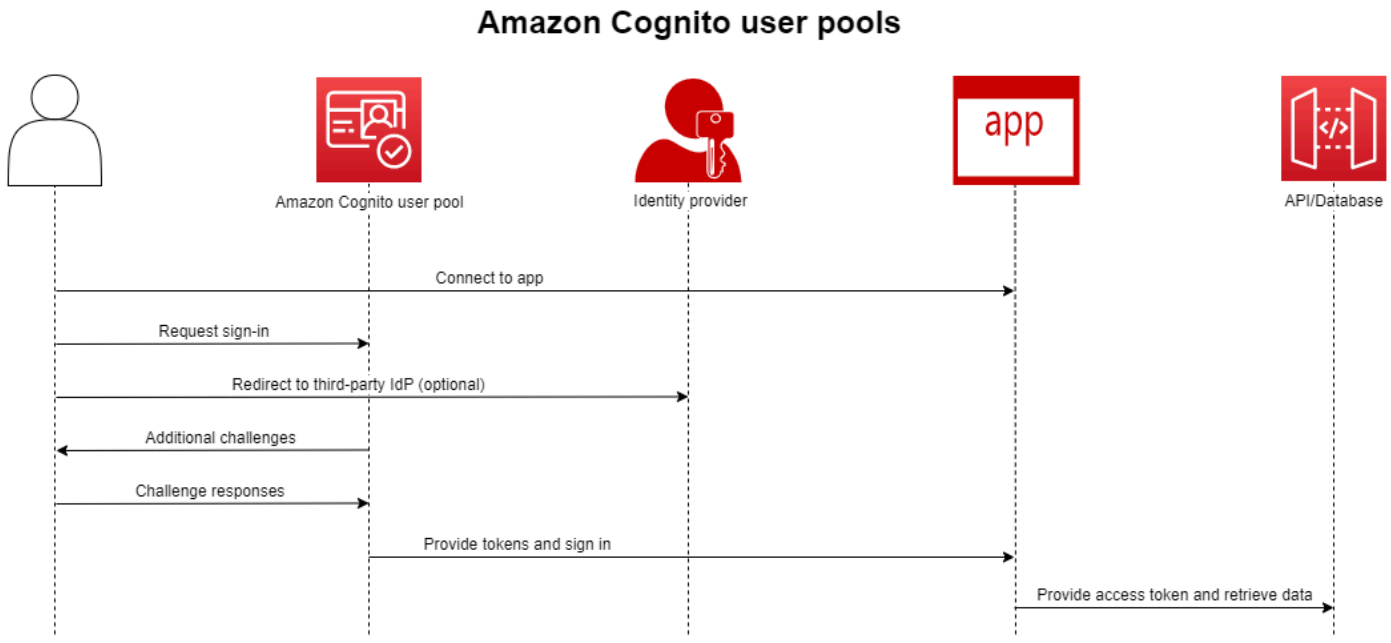
Federate through a social or SAML/OIDC provider. A user pool brokers the sign-in and issues your app a standard set of tokens, or an identity pool exchanges the external token directly for AWS credentials. See Authenticate with a third party and access AWS services with an identity pool.

Authorize an application or device with no human user

Machine-to-machine (M2M) authorization issues an access token to an app, service, or device that authenticates as itself with the `client_credentials` grant, with no human user in the loop. See Authorize machine-to-machine (M2M) applications.

For the full set of scenarios with architecture diagrams, see Common Amazon Cognito scenarios.

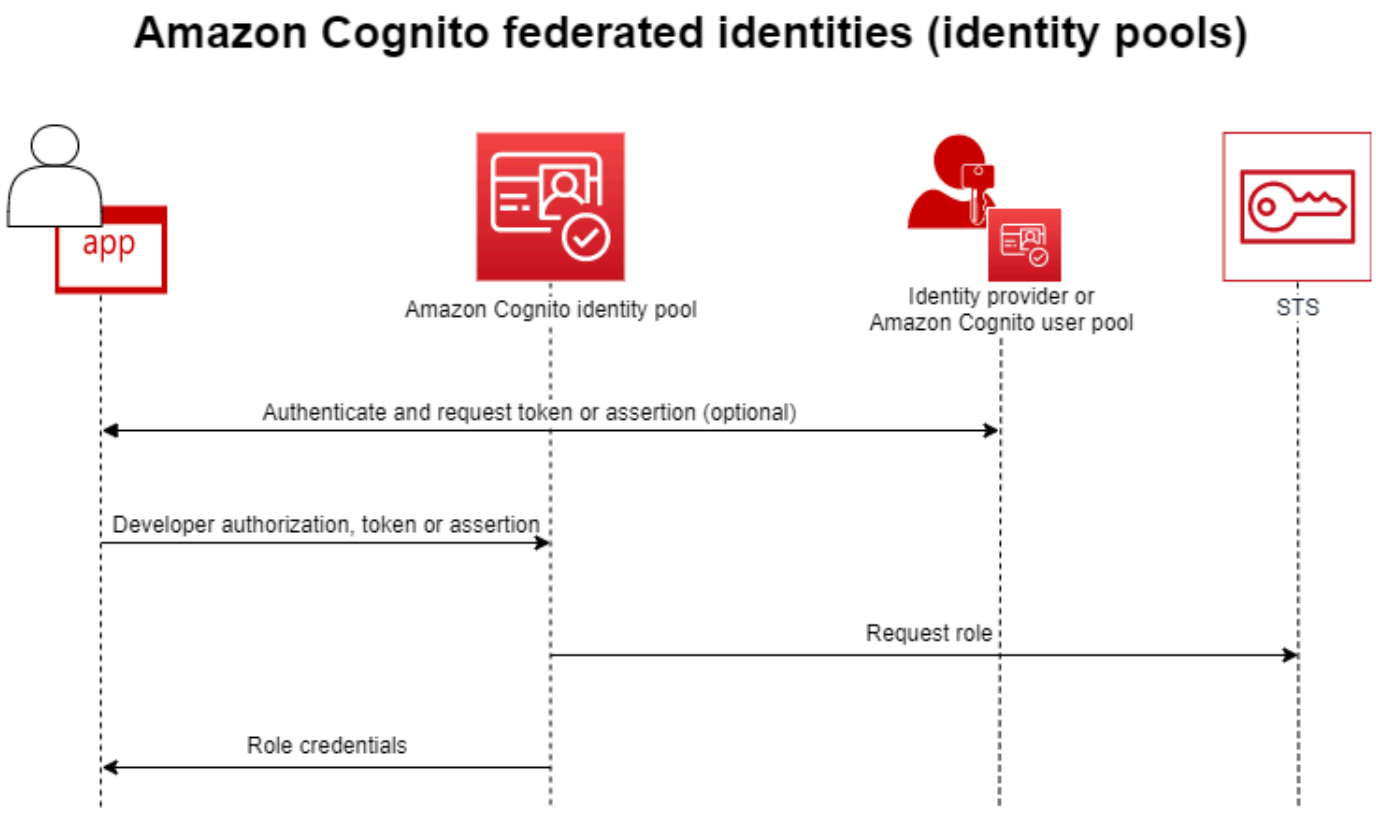
User pools



Create a user pool when you want to authenticate and authorize users to your app or API. User pools are a user directory with both self-service and administrator-driven user creation, management, and authentication. Your user pool can be an independent directory and OIDC identity provider (IdP), and an intermediate service provider (SP) to third-party providers of workforce and customer identities. You can provide single sign-on (SSO) in your app for your organization's workforce identities in SAML 2.0 and OIDC IdPs with user pools. You can also provide SSO in your app for your organization's customer identities in the public OAuth 2.0 identity stores Amazon, Google, Apple and Facebook. For more information about customer identity and access management (CIAM), see What is CIAM?.

User pools don't require integration with an identity pool. From a user pool, you can issue authenticated JSON web tokens (JWTs) directly to an app, a web server, or an API.

Identity pools



Set up an Amazon Cognito identity pool when you want to authorize authenticated or anonymous users to access your AWS resources. An identity pool issues AWS credentials for your app to serve resources to users. You can authenticate users with a trusted identity provider, like a user pool or

a SAML 2.0 service. It can also optionally issue credentials for guest users. Identity pools use both role-based and attribute-based access control to manage your users' authorization to access your AWS resources.

Identity pools don't require integration with a user pool. An identity pool can accept authenticated claims directly from both workforce and consumer identity providers.

An Amazon Cognito user pool and identity pool used together

In the diagram that begins this topic, you use Amazon Cognito to authenticate your user and then grant them access to an AWS service.

1. Your app user signs in through a user pool and receives OAuth 2.0 tokens.
2. Your app exchanges a user pool token with an identity pool for temporary AWS credentials that you can use with AWS APIs and the AWS Command Line Interface (AWS CLI).
3. Your app assigns the credentials session to your user, and delivers authorized access to AWS services like Amazon S3 and Amazon DynamoDB.

For more examples that use identity pools and user pools, see Common Amazon Cognito scenarios.

In Amazon Cognito, the *security of the cloud* obligation of the shared responsibility model is compliant with SOC 1-3, PCI DSS, ISO 27001, and is HIPAA-BAA eligible. You can design your *security in the cloud* in Amazon Cognito to be compliant with SOC1-3, ISO 27001, and HIPAA-BAA, but not PCI DSS. For more information, see AWS services in scope. See also Regional data considerations.

Features of Amazon Cognito

User pools

An Amazon Cognito user pool is a user directory. With a user pool, your users can sign in to your web or mobile app through Amazon Cognito, or federate through a third-party IdP. Federated and local users have a user profile in your user pool.

Local users are those who signed up or you created directly in your user pool. You can manage and customize these user profiles in the AWS Management Console, an AWS SDK, or the AWS Command Line Interface (AWS CLI).

Amazon Cognito user pools accept tokens and assertions from third-party IdPs, and collect the user attributes into a JWT that it issues to your app. You can standardize your app on one set of JWTs while Amazon Cognito handles the interactions with IdPs, mapping their claims to a central token format.

An Amazon Cognito user pool can be a standalone IdP. Amazon Cognito draws from the OpenID Connect (OIDC) standard to generate JWTs for authentication and authorization. When you sign in local users, your user pool is authoritative for those users. You have access to the following features when you authenticate local users.

- Implement your own web front-end that calls the Amazon Cognito user pools API to authenticate, authorize, and manage your users.
- Set up multi-factor authentication (MFA) for your users. Amazon Cognito supports time-based one-time password (TOTP) and SMS message MFA.
- Secure against access from user accounts that are under malicious control.
- Create your own custom multi-step authentication flows.
- Look up users in another directory and migrate them to Amazon Cognito.

An Amazon Cognito user pool can also fulfill a dual role as a service provider (SP) to your IdPs, and an IdP to your app. Amazon Cognito user pools can connect to consumer IdPs like Facebook and Google, or workforce IdPs like Okta and Active Directory Federation Services (ADFS).

With the OAuth 2.0 and OpenID Connect (OIDC) tokens that an Amazon Cognito user pool issues, you can

- Accept an ID token in your app that authenticates a user, and provides the information that you need to set up the user's profile
- Accept an access token in your API with the OIDC scopes that authorize your users' API calls.
- Retrieve AWS credentials from an Amazon Cognito identity pool.

Features of Amazon Cognito user pools

Feature	Description
OIDC identity provider	Issue ID tokens to authenticate users

Authorization server	Issue access tokens to authorize user access to APIs
SAML 2.0 service provider	Transform SAML assertions into ID and access tokens
OIDC relying party	Transform OIDC tokens into ID and access tokens
Social provider relying party	Transform ID tokens from Apple, Facebook, Amazon, or Google to your own ID and access tokens
Authentication frontend service	Sign up, manage, and authenticate users with managed login
API support for your own UI	Create, manage and authenticate users through authentication API requests in supported AWS SDKs[1]
Multi-factor authentication	Use SMS messages, TOTPs, or your user's device as an additional authentication factor[1]
Security monitoring & response	Secure against malicious activity and insecure passwords[1]
Customize authentication flows	Build your own authentication mechanism, or add custom steps to existing flows[2]
Groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools
Customize tokens	Customize your ID and access tokens with new, modified, and suppressed claims
Customize user attributes	Assign values to user attributes and add your own custom attributes

[1] Feature is unavailable to federated users.

[2] Feature is unavailable to federated and managed login users.

For more information about user pools, see Create a new application in the Amazon Cognito console and the Amazon Cognito user pools API reference.

Identity pools

An identity pool is a collection of unique identifiers, or identities, that you assign to your users or guests and authorize to receive temporary AWS credentials. When you present proof of authentication to an identity pool in the form of the trusted claims from a SAML 2.0, OpenID Connect (OIDC), or OAuth 2.0 social identity provider (IdP), you associate your user with an identity in the identity pool. The token that your identity pool creates for the identity can retrieve temporary session credentials from AWS Security Token Service (AWS STS).

To complement authenticated identities, you can also configure an identity pool to authorize AWS access without IdP authentication. You can offer custom proof of authentication with Developer-authenticated identities. You can also grant temporary AWS credentials to guest users, with unauthenticated identities.

With identity pools, you have two ways to integrate with IAM policies in your AWS account. You can use these two features together or individually.

Role-based access control

When your user passes claims to your identity pool, Amazon Cognito chooses the IAM role that it requests. To customize the role's permissions to your needs, you apply IAM policies to each role. For example, if your user demonstrates that they are in the marketing department, they receive credentials for a role with policies tailored to marketing department access needs. Amazon Cognito can request a default role, a role based on rules that query your user's claims, or a role based on your user's group membership in a user pool. You can also configure the role trust policy so that IAM trusts only your identity pool to generate temporary sessions.

Attributes for access control

Your identity pool reads attributes from your user's claims, and maps them to principal tags in your user's temporary session. You can then configure your IAM resource-based policies to allow or deny access to resources based on IAM principals that carry the session tags from your identity pool. For example, if your user demonstrates that they are in the marketing department, AWS STS tags their

session `Department: marketing`. Your Amazon S3 bucket permits read operations based on an aws:PrincipalTag condition that requires a value of `marketing` for the `Department` tag.

Features of Amazon Cognito identity pools

Feature	Description
Amazon Cognito user pool relying party	Exchange an ID token from your user pool for web identity credentials from AWS STS
SAML 2.0 service provider	Exchange SAML assertions for web identity credentials from AWS STS
OIDC relying party	Exchange OIDC tokens for web identity credentials from AWS STS
Social provider relying party	Exchange OAuth tokens from Amazon, Facebook, Google, Apple, and Twitter for web identity credentials from AWS STS
Custom relying party	With AWS credentials, exchange claims in any format for web identity credentials from AWS STS
Unauthenticated access	Issue limited-access web identity credentials from AWS STS without authentication
Role-based access control	Choose an IAM role for your authenticated user based on their claims, and configure your roles to only be assumed in the context of your identity pool
Attribute-based access control	Convert claims into principal tags for your AWS STS temporary session, and use IAM policies to filter resource access based on principal tags

For more information about identity pools, see Getting started with Amazon Cognito identity pools and the Amazon Cognito identity pools API reference.

Amazon Cognito user pools and identity pools comparison

Feature	Description	User pools	Identity pools
OIDC identity provider	Issue OIDC ID tokens to authenticate app users	✓	
User directory	Store user profiles for authentication	✓	
Authorize API access	Issue access tokens to authorize user access to APIs (including user profile self-service API operations), databases, and other resources that accept OAuth scopes	✓	
IAM web identity authorization	Generate tokens that you can exchange with AWS STS for temporary AWS credentials		✓
SAML 2.0 service provider & OIDC identity provider	Issue customized OIDC tokens based on claims from a SAML 2.0 identity provider	✓	
OIDC relying party & OIDC identity provider	Issue customized OIDC tokens based on claims from an OIDC identity provider	✓	

OAuth 2.0 relying party & OIDC identity provider	Issue customized OIDC tokens based on scopes from OAuth 2.0 social providers like Apple and Google	✓	
SAML 2.0 service provider & credentials broker	Issue temporary AWS credentials based on claims from a SAML 2.0 identity provider		✓
OIDC relying party & credentials broker	Issue temporary AWS credentials based on claims from an OIDC identity provider		✓
Social provider relying party & credentials broker	Issue temporary AWS credentials based on JSON web tokens from developer applications with social providers like Apple and Google		✓
Amazon Cognito user pool relying party & credentials broker	Issue temporary AWS credentials based on JSON web tokens from Amazon Cognito user pools		✓
Custom relying party & credentials broker	Issue temporary AWS credentials to arbitrary identities, authorized by developer IAM credentials		✓

Authentication frontend service	Sign up, manage, and authenticate users with managed login	✓
API support for your own authentication UI	Create, manage and authenticate users through API requests in supported AWS SDKs[1]	✓
MFA	Use SMS messages, TOTPs, or your user's device as an additional authentic ation factor[1]	✓
Security monitoring & response	Protect against malicious activity and insecure passwords[1]	✓
Customize authentic ation flows	Build your own authentication mechanism, or add custom steps to existing flows[1]	✓
User groups	Create logical groupings of users, and a hierarchy of IAM role claims when you pass tokens to identity pools	✓
Customize tokens	Customize your ID and access tokens with new, modified, and suppressed claims and scopes	✓

AWS WAF web ACLs	Monitor and control requests to your authentication front end with AWS WAF	✓	
Customize user attributes	Assign values to user attributes and add your own custom attributes	✓	
Unauthenticated access	Issue limited-a ccess web identity credentials from AWS STS without authentication		✓
Role-based access control	Choose an IAM role for your authentic ated user based on their claims, and configure your role trust to limit access to web identity users		✓
Attribute-based access control	Transform user claims into principal tags for your AWS STS temporary session, and use IAM policies to filter resource access based on principal tags		✓

[1] Feature is not available to federated users.

Getting started with Amazon Cognito

For example user pool applications, see Create a new application in the Amazon Cognito console.

For an introduction to identity pools, see Getting started with Amazon Cognito identity pools.

For links to guided setup experiences with user pools and identity pools, see Guided setup options for Amazon Cognito.

To get started with an AWS SDK, see AWS Developer Tools. For developer resources specific to Amazon Cognito, see Amazon Cognito developer resources.

To use Amazon Cognito, you need an AWS account. For more information, see Getting started with AWS.

Regional availability

Amazon Cognito is available in multiple AWS Regions worldwide. In each Region, Amazon Cognito is distributed across multiple Availability Zones. These Availability Zones are physically isolated from each other, but are united by private, low-latency, high-throughput, and highly redundant network connections. These Availability Zones enable AWS to provide services, including Amazon Cognito, with very high levels of availability and redundancy, while also minimizing latency.

To see if Amazon Cognito is currently available in any AWS Region, see AWS Services by Region.

To learn about regional API service endpoints, see AWS regions and endpoints in the *Amazon Web Services General Reference*.

To learn more about the number of Availability Zones that are available in each Region, see AWS global infrastructure.

Pricing for Amazon Cognito

For information about Amazon Cognito pricing, see Amazon Cognito pricing.

Common Amazon Cognito terms and concepts

Amazon Cognito provides credentials for web and mobile apps. It draws from and builds on terms that are common in *identity and access management*. Many guides to universal identity and access terms are available. Some examples are:

- Terminology in the IDPro Body of Knowledge
- AWS Identity Services
- Glossary from NIST CSRC

The following lists describe terms that are unique to Amazon Cognito or have a specific context in Amazon Cognito.

Topics

- General
- User pools
- Identity pools

General

The terms in this list aren't specific to Amazon Cognito and are widely recognized among identity and access management practitioners. The following isn't an exhaustive list of terms, but a guide to their specific Amazon Cognito context in this guide.

Access token

A JSON web token (JWT) that contains information about an entity's authorization to access information systems.

App, application

Typically, a mobile application. In this guide, *app* is often a shorthand for a web application or mobile app that connects to Amazon Cognito.

Attribute-based access control (ABAC)

A model where an app determines access to resources based on the properties of a user, like their job title or department. Amazon Cognito tools to enforce ABAC include ID tokens in user pools and principal tags in identity pools.

Authentication

The process of establishing an authentic identity for the purpose of access to an information system. Amazon Cognito accepts proof of authentication from third-party identity providers, and also serves as a provider of authentication to software applications.

Authorization

The process of granting permissions to a resource. User pool access tokens contain information that applications can use to permit users and systems to access resources.

Authorization server

An OAuth or OpenID Connect (OIDC) system that generates JSON web tokens. The Amazon Cognito user pools managed authorization server is the authorization-server component of the two authentication and authorization methods in user pools. User pools also support API challenge-response flows in SDK authentication.

Confidential app, server-side app

An application that users connect to remotely, with code on an application server and access to secrets. This is typically a web application.

Identity provider (IdP)

A service that stores and verifies user identities. Amazon Cognito can request authentication from external providers and be an IdP to apps.

JSON web token (JWT)

A JSON-formatted document that contains claims about an authenticated user. ID tokens authenticate users, access tokens authorize users, and refresh tokens update credentials. Amazon Cognito receives tokens from external providers and issues tokens to apps or AWS STS.

Machine-to-machine (M2M) authorization

The process of authorizing requests to API endpoints for non-user-interactive machine entities, like a webserver application tier. User pools serve M2M authorization in client-credentials grants with OAuth 2.0 scopes in access tokens.

Multi-factor authentication (MFA)

The requirement that users provide additional authentication after providing their username and password. Amazon Cognito user pools have MFA features for local users.

OAuth 2.0 (social) provider

An IdP to a user pool or identity pool that provides JWT access and refresh tokens. Amazon Cognito user pools automate interactions with social providers after users authenticate.

OpenID Connect (OIDC) provider

An IdP to a user pool or identity pool that extends the OAuth specification to provide ID tokens. Amazon Cognito user pools automate interactions with OIDC providers after users authenticate.

Passkey, WebAuthn

A form of authentication where cryptographic keys, or passkeys, on a user's device provides their proof of authentication. Users verify that they are present with biometric or PIN code mechanisms in a hardware or software authenticator. Passkeys are phishing-resistent and bound to specific websites/apps, offering a secure passwordless experience. Amazon Cognito user pools support sign-in with passkeys.

Passwordless

A form of authentication where a user doesn't have to enter a password. Methods of passwordless sign-in include one-time passwords (OTPs) sent to email addresses and phone numbers, and passkeys. Amazon Cognito user pools support sign-in with OTPs and passkeys.

Public app

An application that is self-contained on a device, with code stored locally and no access to secrets. This is typically a mobile app.

Resource server

An API with access control. Amazon Cognito user pools also use *resource server* to describe the component that defines the configuration for interacting with an API.

Role-based access control (RBAC)

A model that grants access based on a user's functional designation. Amazon Cognito identity pools implement RBAC with differentiation between IAM roles.

Service provider (SP), relying party (RP)

An application that relies on an IdP to assert that users are trustworthy. Amazon Cognito acts as an SP to external IdPs, and as an IdP to app-based SPs.

SAML provider

An IdP to a user pool or identity pool that generates digitally signed assertion documents that your user passes to Amazon Cognito.

Universally Unique Identifier (UUID)

A 128-bit label that is applied to an object. Amazon Cognito UUIDs are unique per user pool or identity pool, but don't conform to a specific UUID format, including RFC UUID. You shouldn't strictly validate the format of Amazon Cognito identifiers.

User directory

A collection of users and their attributes that serves that information to other systems. Amazon Cognito user pools are user directories, and also tools for consolidation of users from external user directories.

User pools

When you see the terms in the following list in this guide, they refer to a specific feature or configuration of user pools.

Adaptive authentication

A feature of advanced security that detects potential malicious activity and applies additional security to user profiles.

App client

A component that defines the settings for a user pool as an IdP to one app.

Callback URL, redirect URI, return URL

A setting in an app client and a parameter in requests to the user pool's authorization server. The callback URL is the initial destination for authenticated users in your app.

Choice-based authentication

A form of API authentication with users pools where each user has a set of choices for sign-in available to them. Their choices might include username and password with or without MFA, passkey sign-in, or passwordless sign-in with email or SMS message one-time passwords. Your application can shape the choice process for users by requesting a list of authentication options or by declaring a preferred option.

Compare with client-based authentication.

Client-based authentication

A form of authentication with the user pools API and application back ends built with AWS SDKs. In declarative authentication, your application determines independently the login type that a user should perform and requests that type up front.

Compare with choice-based authentication.

Compromised credentials

A feature of advanced security that detects user passwords that attackers might know, and applies additional security to user profiles.

Confirmation

The process that determines that the prerequisites have been met to permit a new user to sign in. Confirmation is typically done through email address or phone number verification.

Custom authentication

An extension of authentication processes with Lambda triggers that define additional user challenges and responses.

Device authentication

An authentication process that replaces MFA with sign-in that uses the ID of a trusted device.

Domain, user pool domain

A web domain that hosts your managed login pages in AWS. You can set up DNS in a domain that you own or use an identifying subdomain prefix in a domain that AWS owns.

Essentials plan

The feature plan with the latest developments in user pools. The Essentials plan doesn't include the automated-learning security features in the Plus plan.

External provider, third-party provider

An IdP that has a trust relationship with a user pool. User pools serve as an intermediate entity between external providers and your application, managing authentication processes with SAML 2.0, OIDC, and social providers. User pools consolidate external-provider authentication outcomes into a single IdP so that your applications can process many users with a single OIDC relying-party library.

Feature plan

The group of features that you can select for a user pool. Feature plans have differing costs in your AWS bill. New user pools default to the Essentials plan.

Current plans

- Lite plan
- Essentials plan
- Plus plan

Federated user, external user

A user in a user pool who was authenticated by an external provider.

Hosted UI (classic), hosted UI pages

The early version of the authentication front end, relying party, and identity provider services on your user pool domain. The hosted UI has a basic set of features and a simplified look and feel. You can apply Hosted UI branding with the upload of a logo-image file and a file with a predetermined set of CSS styles. Compare to managed login.

Lambda trigger

A function in AWS Lambda that a user pool can automatically invoke at key points in user authentication processes. You can use Lambda triggers to customize authentication outcomes.

Local user

A user profile in the user pool user directory that wasn't created by authentication with an external provider.

Linked user

A user from an external provider whose identity is merged with a local user.

Lite plan

The feature plan with the features that originally launched with user pools. The Lite plan doesn't include the new features in the Essentials plan or the automated-learning security features in the Plus plan.

Managed authorization server, hosted UI authorization server, authorization server

A component of managed login that hosts services for interaction with IdPs and apps on your user pool domain. The hosted UI differs from managed login in the user-interactive features it offers, but has the same authorization-server capabilities.

Managed login, managed login pages

A set of webpages on your user pool domain that host services for user authentication. These services include functions for operating as an IdP, a relying party for third-party IdPs, and a server of a user-interactive authentication UI. When you set up a domain for your user pool, Amazon Cognito brings all managed login pages online.

Your application import OIDC libraries that invoke users' browsers and direct them to the managed login UI for sign-up, sign-in, password management, and other authentication operations. After authentication, the OIDC libraries can process the outcome of the authentication request.

Managed login authentication

Sign-in with the services on your user pool domain, done with user-interactive browser pages or HTTPS API requests. Applications handle managed login authentication with OpenID Connect (OIDC) libraries. This process includes sign-in with external providers, local-user sign-in with interactive managed login pages, and M2M authorization. Authentication with the classic hosted UI also fall under this term.

Compare to AWS SDK authentication.

Plus plan

The feature plan with the latest developments and advanced security features in user pools.

SDK authentication, AWS SDK authentication

A set of authentication and authorization API operations that you can add to your application back end with an AWS SDK. This authentication model requires your own custom-built login mechanism. The API can sign in local users and linked users.

Compare to managed login authentication.

Threat protection, advanced security features

In user pools, threat protection refers to technologies that are designed to mitigate threats to your authentication and authorization mechanisms. Adaptive authentication, compromised-credentials detection, and IP address blocklists are under the category of threat protection.

Token customization

The outcome of a pre token generation Lambda trigger that modifies a user's ID or access token at runtime.

User pool, Amazon Cognito identity provider, `cognito-idp`, Amazon Cognito user pools

An AWS resource with authentication and authorization services for applications that work with OIDC IdPs.

Verification

The process of confirming that a user owns an email address or phone number. A user pool sends a code to a user who has entered a new email address or phone number. When they submit the code to Amazon Cognito, they verify their ownership of the message destination and can receive additional messages from the user pool. Also, see confirmation.

User profile, user account

An entry for a user in the user directory. All users, including those from third-party IdPs, have a profile in their user pool.

Identity pools

When you see the terms in the following list in this guide, they refer to a specific feature or configuration of identity pools.

Attributes for access control

An implementation of attribute-based access control in identity pools. Identity pools apply user attributes as tags to user credentials.

Basic (classic) authentication

An authentication process where you can customize the request for user credentials.

Developer authenticated identities

An authentication process that authorizes identity pool user credentials with developer credentials.

Developer credentials

The IAM API keys of an identity pool administrator.

Enhanced authentication

An authentication flow that selects an IAM role and applies principal tags according to the logic that you define in your identity pool.

Identity

A UUID that links an app user and their user credentials to their profile in an external user directory that has a trust relationship with an identity pool.

Identity pool, Amazon Cognito federated identities, Amazon Cognito identity, `cognito-identity`

An AWS resource with authentication and authorization services for applications that use temporary AWS credentials.

Unauthenticated identity

A user who has not signed in with an identity pool IdP. You can permit users to generate limited user credentials for a single IAM role before they authentication.

User credentials

Temporary AWS API keys that users receive after identity pool authentication.

Getting started with AWS

Before you start working with Amazon Cognito, set yourself up with some required AWS resources. If you can already sign in to an AWS account, you can skip this section. Continue reading if you are looking for information about signing up and signing in with AWS credentials. After you have credentials with sufficient AWS Identity and Access Management (IAM) permissions, you can get started with user pools and identity pools.

Sign up for an AWS account

To get started with AWS, you need an AWS account. For information about creating an AWS account, see Getting started with an AWS account in the *AWS Account Management Reference Guide*.

Getting started with Amazon Cognito

This chapter gets you started with user pool authentication—creating your first user pool and signing in your first user. Amazon Cognito also offers identity pools, which exchange a user's identity for temporary AWS credentials; if that is what you need, see *Exchanging identities for AWS credentials*. If you are not sure which component you need, see the section called "Which parts of Amazon Cognito do you need?".

> **Note**
>
> You need an AWS account to complete the walkthrough and quickstarts in this chapter. If you don't have one, see the section called "Getting started with AWS".

You have an application that requires authentication and access control. You can work within the OpenID Connect (OIDC) framework for single sign-on (SSO). Amazon Cognito has tools for handling the logic of authentication in the application back end with an AWS SDK, and for invoking a browser in your client to access a managed authorization server.

The Amazon Cognito console guides you through the creation of a user pool from the view of your preferred application framework. From there, you can continue on to add features like federated sign-in with external social or SAML 2.0 identity providers (IdPs). The application models in the Amazon Cognito console lean on the addition of OIDC libraries to your project and invoking a browser.

As you work to expand your feature set and incorporate more components of Amazon Cognito, read the Amazon Cognito user pools chapter for full descriptions of everything you can do with user pools.

The examples in this chapter and in the Amazon Cognito console demonstrate a basic integration of application resources with Amazon Cognito user pools. Later, you can adjust your user pool to use more of the options that are available to you. Then you can update your application to adopt new features and interact with IdPs.

If you don't want to use the managed login pages, you can create an application with custom-built authentication interfaces using an AWS SDK or AWS Amplify. Applications that you build in this way interact with the user pools API and are suitable only for authenticating local users. Continue learning about this authentication model at Other application options.

Your first sign-in: an end-to-end walkthrough

This page is a map, not a replacement for the detailed procedures it links to. It shows how the setup steps connect so that you can see the whole journey before you dive into any single step. Each numbered step links to the authoritative procedure.

> **Note**
>
> This walkthrough uses a user pool with managed login, the most common starting point for an application that signs in users. *Managed login* is the current Amazon Cognito-hosted sign-in UI, with a visual branding editor; the earlier *classic hosted UI* is a simpler alternative. You choose one or the other as the branding version on your user pool domain, and that choice applies to all app clients that use the domain. Managed login requires the *Essentials* or *Plus* feature plan—on the *Lite* plan, the classic hosted UI is your only hosted option. For the differences, see the section called "Managed login" and the section called "User pool feature plans". If your goal is temporary AWS credentials for an existing identity source instead, start with the section called "Getting started with identity pools".

From empty account to validated token

1. **Create a user pool.** A user pool is your user directory and the identity provider that issues tokens. See the section called "Your first app and user pool".
2. **Add a domain.** Managed login pages are served from a user pool domain—either an Amazon Cognito prefix domain or your own custom domain. Without a domain, there is no hosted sign-in page to send users to. See the section called "Configuring a domain".
3. **Create an app client.** An app client represents your application to the user pool. It carries the settings that control sign-in, such as the allowed callback URLs and the sign-in methods. The branding version (managed login or classic hosted UI) is set on the domain, not the app client. See the section called "App clients".
4. **Send the user to managed login to sign in.** With a domain and an app client in place, your application redirects users to the managed login pages to sign up and sign in. Amazon Cognito returns your user to your callback URL with an authorization code. See the section called "Managed login".
5. **Exchange the code for tokens.** Your application exchanges the authorization code at the token endpoint for an ID token, an access token, and a refresh token. See the section called "Token endpoint".

6. **Validate the token.** Your application (or your API) verifies the token's signature and claims before trusting it. See the section called “Verifying a JSON Web Token”.

At this point you have a signed-in user and a validated token. To authorize that token against your own API or backend—the most common next step—see the section called “Accessing resources”. If instead you need to grant the user temporary AWS credentials to call AWS services such as Amazon S3 directly from your app, continue to the section called “Amazon Cognito identity pools”.

Create a new application in the Amazon Cognito console

User pools add authentication options to software applications. For the easiest getting-started experience, step into the Amazon Cognito console and follow the instructions there. The creation process there guides you not only through setup of user pool resources, but through setting up the initial pieces of your application.

When you're ready to begin, navigate to the Amazon Cognito console and select the button to create a new user pool. The setup process will guide you through your configuration and programming-language options.

Additional resources for authentication concepts

- Authentication with Amazon Cognito user pools
- Understanding API, OIDC, and managed login pages authentication
- How authentication works with Amazon Cognito
- Integrating Amazon Cognito authentication and authorization with web and mobile apps

To create Amazon Cognito resources for your application

1. Navigate to the Amazon Cognito console. To assign permissions to your IAM principal so that they can create and manage Amazon Cognito resources, refer to AWS managed policies for Amazon Cognito. The `AmazonCognitoPowerUser` policy is sufficient for the creation of user pools.
2. Select **Create user pool** from the **User pools** menu, or select **Get started for free in less than five minutes**.
3. Under **Define your application**, choose the **Application type** that best fits the application scenario that you want to create authentication and authorization services for.

4. In **Name your application**, enter a descriptive name or proceed with the default name.
5. You must make some basic choices under **Configure options** that support settings that you can't change after you create your user pool.

   a. Under **Options for sign-in identifiers**, tell us how you want to identify users when they sign in. You can prefer user-generated usernames, email addresses, or phone numbers. You can also allow a combination of multiple options. Amazon Cognito accepts the options that you configure here in the username field of managed login sign-in forms.

   b. Under **Required attributes for sign-up**, tell us what user information you want to collect when users register for a new account. In managed login pages, Amazon Cognito presents prompts for all required attributes.

      Options for sign-in identifiers influences your required attributes. **Username** requires email or phone attributes for each user so that they can receive a password-reset code in an email or SMS message. **Email** requires the email attribute, and **Phone number** requires the phone number attribute.
6. Under **Add a return URL**, enter a redirect path to your application for after users complete authentication. This location should be a route in your application that uses OpenID Connect (OIDC) libraries to process user-authentication outcomes. An example of a return URL for a test application is `https://localhost:3000/callback`. In the example NodeJS application in the Amazon Cognito console, this route employs openid-client to collect the access token and redeem it for user information. You'll be able to browse examples for your development platform after you create your resources.
7. Choose **Create your application**. Amazon Cognito creates a user pool and app client with default settings for your application type. You can configure additional options like external identity providers and multi-factor authentication (MFA) after you create your initial resources.
8. On the **Set up your application** page, you can immediately get code examples for your application. To explore your new user pool, scroll down and select **Go to overview**.
9. To add more applications in the same user pool, navigate to the **App clients** menu and add a new app client. This will repeat the process of application-focused creation, but only add a new app client to the existing user pool.

After you create a user pool and one or more app clients with this process, you can start testing authentication operations with managed login. These quick-start options are open to public self sign-up. We recommend that you create a testing environment with the console process, then move your finalized design to production. Spend time familiarizing yourself with the capabilities of

Amazon Cognito. Then, to move to production workloads, craft custom configurations and deploy them with automation tools like AWS CloudFormation and the AWS Cloud Development Kit (AWS CDK).

Amazon Cognito makes some default configurations in this process that you can't reverse. For more information about user pool settings that you can't change and those options that you can choose in the console, see Updating user pool and app client configuration.

Setting	Effect	How to change	More information
Client secret	Requires a client secret hash in authentication requests.	Create a new app client with a **Traditional web application** or **Machine-to-machine application** profile.	Application-specific settings with app clients
Preferred username	User pool doesn't accept the `preferred_username` attribute as an alias.	Create a user pool programmatically with an AWS SDK.	Customizing sign-in attributes
Case sensitivity	User pool usernames are case insensitive, for example `JohnD` is considered to be the same user as `johnd`.	Create a user pool programmatically with an AWS SDK.	User pool case sensitivity

Other application options

You might have an existing application UI that you want to integrate with Amazon Cognito authentication. You might even have your own existing authentication pages with a less-functional directory setup than Amazon Cognito user pools. You can add or replace an authentication component to an application of this type with Amazon Cognito integrations in AWS SDKs for a variety of programming languages. Some examples follow.

If you create a user pool for this purpose in the Amazon Cognito console, it might not be necessary to have a user pool domain that hosts interactive sign-in pages and OpenID Connect (OIDC) services. The process of user pool creation in the console automatically generates a domain for you. You can delete this domain from the **Domain** tab of your user pool. Other options include programmatic creation of Amazon Cognito resources for you application with API requests in AWS SDKs and with the automated-setup options in the AWS Amplify CLI. For more information, see Integrating Amazon Cognito authentication and authorization with web and mobile apps.

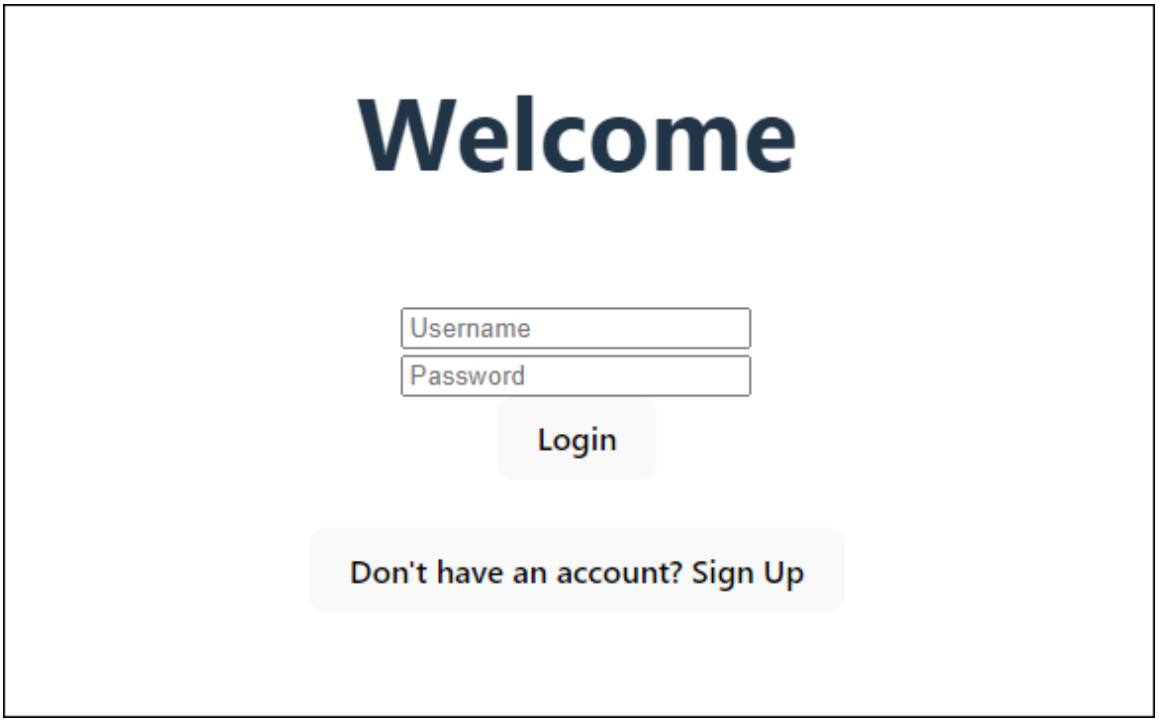
Topics

- Set up an example React single page application
- Set up an example Android app with Flutter

Set up an example React single page application

In this tutorial, you'll create a React single page application where you can test user sign-up, confirmation, and sign-in. React is a JavaScript-based library for web and mobile apps, with a focus on the user interface (UI). This example application demonstrates some basic functions of Amazon Cognito user pools. If you're already experienced in web app development with React, download the example app from GitHub.

The following screenshot is of the initial authentication page in the application that you'll create.



To set up this application, your user pool must meet the following requirements:

- Users can sign in with their email address. **Cognito user pool sign-in options**: **Email**.
- Usernames are case insensitive. **User name requirements**: **Make user name case sensitive** is not selected.
- Multi-factor authentication (MFA) isn't required. **MFA enforcement**: **Optional MFA**.
- Your user pool verifies attributes for user-profile confirmation with an email message. **Attributes to verify**: **Send email message, verify email address**.
- Email is the only required attribute. **Required attributes**: **email**.
- Users can sign themselves up in your user pool. **Self-registration**: **Enable self-registration** is selected.
- Your initial app client is a public client that permits sign-in with username and password. **App type**: **Public client**, **Authentication flows**: `ALLOW_USER_PASSWORD_AUTH`.

Create an application

To build this application, you must set up a developer environment. The developer environment requirements are:

1. Node.js is installed and updated.
2. Node package manager (npm) is installed and updated to at least version 10.2.3.
3. The environment is accessible on TCP port 5173 in a web browser.

To create an example React web application

1. Sign in to your developer environment and navigate to the parent directory for your application.

   ```
   cd ~/path/to/project/folder/
   ```

2. Create a new React service.

   ```
   npm create vite@latest frontend-client -- --template react-ts
   ```

3. Clone the `cognito-developer-guide-react-example` project folder from the AWS code examples repository on GitHub.

```
cd ~/some/other/path
```

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

```
cp -r ./aws-doc-sdk-examples/javascriptv3/example_code/cognito-identity-provider/
scenarios/cognito-developer-guide-react-example/frontend-client ~/path/to/project/
folder/
```

4. Navigate to the `src` directory in your project.

```
cd ~/path/to/project/folder/frontend-client/src
```

5. Edit `config.json` and replace the following values:

   a. Replace `YOUR_AWS_REGION` with an AWS Region code. For example: `us-east-1`.

   b. Replace `YOUR_COGNITO_USER_POOL_ID` with the ID of the user pool that you have designated for testing. For example: `us-east-1_EXAMPLE`. The user pool must be in the AWS Region that you entered in the previous step.

   c. Replace `YOUR_COGNITO_APP_CLIENT_ID` with the ID of the app client that you have designated for testing. For example: `1example23456789`. The app client must be in the user pool from the previous step.

6. If you want to access your example application from an IP other than `localhost`, edit `package.json` and change the line `"dev": "vite",` to `"dev": "vite --host 0.0.0.0",`.

7. Install your application.

```
npm install
```

8. Launch the application.

```
npm run dev
```

9. Access the application in a web browser at `http://localhost:5173` or `http://[IP address]:5173`.

10. Sign up a new user with a valid email address.

11. Retrieve the confirmation code from your email message. Enter the confirmation code into the application.
12. Sign in with your username and password.

Creating a React developer environment with Amazon Lightsail

A quick way to get started with this application is to create a virtual cloud server with Amazon Lightsail.

With Lightsail, you can quickly create a small server instance that comes preconfigured with the prerequisites for this example application. You can SSH to your instance with a browser-based client, and connect to the web server at a public or private IP address.

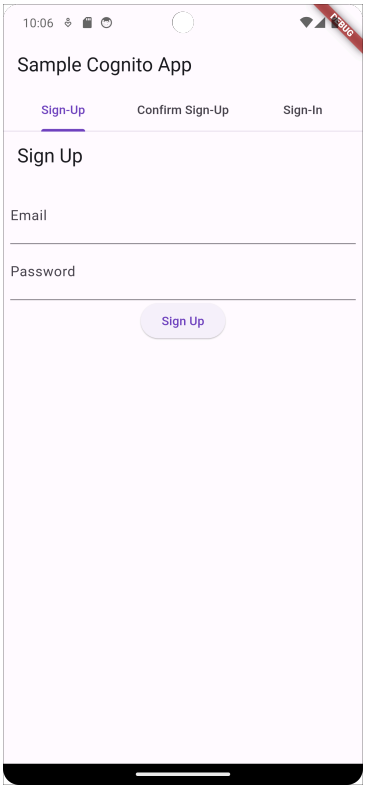
To create a Lightsail instance for this example application

1. Go to the Lightsail console. If prompted, enter your AWS credentials.
2. Choose **Create instance**.
3. For **Select a platform**, choose **Linux/Unix**.
4. For **Select a blueprint**, choose **Node.js**.
5. Under **Identify your instance**, give your development environment a friendly name.
6. Choose **Create instance**.
7. After Lightsail has created your instance, select it and from the **Connect** tab, choose **Connect using SSH**.
8. An SSH session opens in a browser window. Run `node -v` and `npm -v` to confirm that your instance was provisioned with Node.js and the minimum npm version of 10.2.3.
9. Proceed to configure your React application.

Set up an example Android app with Flutter

In this tutorial, you'll create a mobile application in Android Studio where you can emulate a device and test user sign-up, confirmation, and sign-in. This example application creates a basic Amazon Cognito user pools mobile client for Android in Flutter. If you're already experienced in mobile app development with Flutter, download the example app from GitHub.

The following screenshot shows the app running on a virtual Android device.



To set up this application, you user pool must meet the following requirements:

- Users can sign in with their email address. **Cognito user pool sign-in options**: **Email**.
- Usernames are case insensitive. **User name requirements**: **Make user name case sensitive** is not selected.
- Multi-factor authentication (MFA) isn't required. **MFA enforcement**: **Optional MFA**.
- Your user pool verifies attributes for user-profile confirmation with an email message. **Attributes to verify**: **Send email message, verify email address**.
- Email is the only required attribute. **Required attributes**: **email**.
- Users can sign themselves up in your user pool. **Self-registration**: **Enable self-registration** is selected.
- Your initial app client is a public client that permits sign-in with username and password. **App type**: **Public client**, **Authentication flows**: `ALLOW_USER_PASSWORD_AUTH`.

Create an application

To create an example Android app

1. Install Android studio and command-line tools.
2. In Android Studio, install the Flutter plugin.
3. Create a new Android Studio project from the contents of the `cognito_flutter_mobile_app` directory in this example app.
   - Edit `assets/config.json` and replace `<<YOUR USER POOL ID>>` and `<< YOUR CLIENT ID>>` with the IDs of your user pool and app client.
4. Install Flutter.
   a. Add Flutter to your PATH variable.
   b. Accept licenses with the following command.

      ```
      flutter doctor --android-licenses
      ```

   c. Verify your Flutter environment and install any missing components.

      ```
      flutter doctor
      ```

      - If any components are missing, run `flutter doctor -v` to learn how to fix the issue.

   d. Change to the directory of your new Flutter project and install dependencies.
      - Run `flutter pub add amazon_cognito_identity_dart_2`.
   e. Run `flutter pub add flutter_secure_storage`.
5. Create a virtual Android device.
   1. In the Android studio GUI, create a new device with the device manager.
   2. In the CLI, run `flutter emulators --create --name android-device`.
6. Launch your virtual Android device.
   1. In the Android Studio GUI, select the start

      icon next to your virtual device.
   2. In the CLI, run `flutter emulators --launch android-device`.

7. Launch your app on your virtual device.

    1. In the Android Studio GUI, select the deploy

       icon.
    2. In the CLI, run `flutter run`.
8. Navigate to your running virtual device in Android Studio.
9. Sign up a new user with a valid email address.
10. Retrieve the confirmation code from your email message. Enter the confirmation code into the application.
11. Sign in with your username and password.

Add more features and security options to your user pool

After you have followed the tutorials to complete example applications, you can broaden the scope of your user pool implementation. Or, if you didn't create a test application, create a new user pool according to your preferences. You can customize user pool features for other applications or add external identity providers. As you plan your move to put Amazon Cognito user pools in production applications, you can evaluate additional examples and tutorials.

If your next priority is to examine and apply application security options in your user pools, see Security best practices for Amazon Cognito user pools.

Amazon Cognito has feature plans that add functional and security options when you opt in to higher tiers. You can start with the *Lite* plan, add advanced authentication and authorization options with the *Essentials* plan, and add automated-reasoning security guardrails with the *Plus* plan. For more information, see User pool feature plans.

The following are some additional Amazon Cognito user pools features:

- Apply branding to managed login pages
- Adding MFA to a user pool
- Advanced security with threat protection
- Customizing user pool workflows with Lambda triggers
- Using Amazon Pinpoint for user pool analytics

For an overview of Amazon Cognito authentication and authorization models, see How authentication works with Amazon Cognito.

To access other AWS services after a successful user pool authentication, see Accessing AWS services using an identity pool after sign-in.

In addition to using the AWS Management Console and the user pool SDKs, you can also manage your user pools by using the AWS Command Line Interface.

Topics

- Add social sign-in to your user pool
- Add a SAML 2.0 identity provider

Add social sign-in to your user pool

Providing users with the ability to sign in to your application through their existing public, or social, identity providers can improve their authentication experience. Amazon Cognito user pools integrate with popular social identity providers (IdPs) like Facebook, Google, Amazon, and Apple, giving your users convenient sign-in options that they are already familiar with.

When you set up social sign-in, you are giving your users an alternative to creating a dedicated account just for your application. This can improve conversion rates and make the sign-up process more seamless. From the user's perspective, they can apply their existing social credentials to quickly authenticate, without the friction of remembering another username and password.

Configuring a social IdP in your user pool involves a few key steps. You must register your application with the social provider to obtain a client ID and secret. Then you can add the social IdP configuration to your user pool, specifying the scopes that you want to request and the user pool attributes that you want to map from IdP attributes. At runtime, Amazon Cognito handles the token exchange with the provider, maps user attributes, and issues tokens to your application in the shared user pool format.

Register with a social IdP

Before you create a social IdP with Amazon Cognito, you must register your application with the social IdP to receive a client ID and client secret.

To register an app with Facebook

1. Create a developer account with Facebook.

2. Sign in with your Facebook credentials.
3. From the **My Apps** menu, choose **Create New App**.

   If you don't have an existing Facebook app, you will see a different option. Choose **Create App**.
4. On the **Create an app** page, choose a use case for your app, and then choose **Next**.
5. Enter a name for your Facebook app and choose **Create App**.
6. On the left navigation bar, choose **App Settings**, and then choose **Basic**.
7. Note the **App ID** and the **App Secret**. You will use them in the next section.
8. Choose **+ Add platform** from the bottom of the page.
9. On the **Select Platform** screen, select your platforms, and then choose **Next**.
10. Choose **Save changes**.
11. For **App Domains**, enter your user pool domain.

    ```
    https://your_user_pool_domain
    ```
12. Choose **Save changes**.
13. From the navigation bar, choose **Products**, and then choose **Configure** from **Facebook Login**.
14. From the **Facebook Login Configure** menu, choose **Settings**.

    Enter your redirect URL into **Valid OAuth Redirect URIs**. The redirect URL consists of your user pool domain with the `/oauth2/idpresponse` endpoint.

    ```
    https://your_user_pool_domain/oauth2/idpresponse
    ```
15. Choose **Save changes**.

To register an app with Amazon

1. Create a developer account with Amazon.
2. Sign in with your Amazon credentials.
3. You need to create an Amazon security profile to receive the Amazon client ID and client secret.

   Choose **Apps and Services** from the navigation bar at the top of the page, and then choose **Login with Amazon**.

4. Choose **Create a Security Profile**.
5. Enter a **Security Profile Name**, a **Security Profile Description**, and a **Consent Privacy Notice URL**.
6. Choose **Save**.
7. Choose **Client ID** and **Client Secret** to show the client ID and secret. You will use them in the next section.
8. Hover over the gear icon and choose **Web Settings**, and then choose **Edit**.
9. Enter your user pool domain into **Allowed Origins**.

```
https://<your-user-pool-domain>
```

10. Enter your user pool domain with the `/oauth2/idpresponse` endpoint into **Allowed Return URLs**.

```
https://<your-user-pool-domain>/oauth2/idpresponse
```

11. Choose **Save**.

To register an app with Google

For more information about OAuth 2.0 in the Google Cloud platform, see Learn about authentication & authorization in the Google Workspace for Developers documentation.

1. Create a developer account with Google.
2. Sign in to the Google Cloud Platform console.
3. From the top navigation bar, choose **Select a project**. If you already have a project in the Google platform, this menu displays your default project instead.
4. Select **NEW PROJECT**.
5. Enter a name for your product and then choose **CREATE**.
6. On the left navigation bar, choose **APIs and Services**, and then choose **Oauth consent screen**.
7. Enter the app information, an **App domain**, **Authorized domains**, and **Developer contact information**. Your **Authorized domains** must include `amazoncognito.com` and the root of your custom domain. For example: `example.com`. Choose **SAVE AND CONTINUE**.
8. 1. Under **Scopes**, choose **Add or remove scopes**, and then choose, at a minimum, the following OAuth scopes.

    1. `.../auth/userinfo.email`
    2. `.../auth/userinfo.profile`
    3. openid
9. Under **Test users**, choose **Add users**. Enter your email address and any other authorized test users, and then choose **SAVE AND CONTINUE**.
10. Expand the left navigation bar again, choose **APIs and Services**, and then choose **Credentials**.
11. Choose **CREATE CREDENTIALS**, and then choose **OAuth client ID**.
12. Choose an **Application type** and give your client a **Name**.
13. Under **Authorized JavaScript origins**, choose **ADD URI**. Enter your user pool domain.

    ```
    https://<your-user-pool-domain>
    ```
14. Under **Authorized redirect URIs**, choose **ADD URI**. Enter the path to the `/oauth2/idpresponse` endpoint of your user pool domain.

    ```
    https://<your-user-pool-domain>/oauth2/idpresponse
    ```
15. Choose **CREATE**.
16. Securely store the values that Google displays under **Your client ID** and **Your client secret**. Provide these values to Amazon Cognito when you add a Google IdP.

To register an app with Apple

For more information about setting up Sign in with Apple, see Configuring Your Environment for Sign in with Apple in the Apple Developer documentation.

1. Create a developer account with Apple.
2. Sign in with your Apple credentials.
3. On the left navigation bar, choose **Certificates, Identifiers & Profiles**.
4. On the left navigation bar, choose **Identifiers**.
5. On the **Identifiers** page, choose the **+** icon.
6. On the **Register a New Identifier** page, choose **App IDs**, and then choose **Continue**.
7. On the **Select a type** page, choose **App**, and then choose **Continue**.
8. On the **Register an App ID** page, do the following:

1. Under **Description**, enter a description.
2. Under **App ID Prefix**, enter a **Bundle ID**. Make a note of the value under **App ID Prefix**. You will use this value after you choose Apple as your identity provider in Using social identity providers with a user pool.
3. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.
4. On the **Sign in with Apple: App ID Configuration** page, choose to set up the app as either primary or grouped with other App IDs, and then choose **Save**.
5. Choose **Continue**.

9. On the **Confirm your App ID** page, choose **Register**.
10. On the **Identifiers** page, choose the **+** icon.
11. On the **Register a New Identifier** page, choose **Services IDs**, and then choose **Continue**.
12. On the **Register a Services ID** page, do the following:

    1. Under **Description**, enter a description.
    2. Under **Identifier**, enter an identifier. Make a note of this Services ID because you'll need this value after you choose Apple as your identity provider in Using social identity providers with a user pool.
    3. Choose **Continue** and then choose **Register**.
13. Choose the Services ID that you just created from the Identifiers page.

    1. Select **Sign In with Apple**, and then choose **Configure**.
    2. On the **Web Authentication Configuration** page, select the app ID that you created earlier as the **Primary App ID**.
    3. Choose the **+** icon next to **Website URLs**.
    4. Under **Domains and subdomains**, enter your user pool domain without an `https://` prefix.

       ```
       <your-user-pool-domain>
       ```

    5. Under **Return URLs**, enter the path to the `/oauth2/idpresponse` endpoint of your user pool domain.

       ```
       https://<your-user-pool-domain>/oauth2/idpresponse
       ```

    6. Choose **Next**, and then choose **Done**. You don't need to verify the domain.

7. Choose **Continue**, and then choose **Save**.

14. On the left navigation bar, choose **Keys**.
15. On the **Keys** page, choose the **+** icon.
16. On the **Register a New Key** page, do the following:

    1. Under **Key Name**, enter a key name.
    2. Choose **Sign In with Apple**, and then choose **Configure**.
    3. On the **Configure Key** page, select the app ID that you created earlier as the **Primary App ID**. Choose **Save**.
    4. Choose **Continue**, and then choose **Register**.
17. On the **Download Your Key** page, choose **Download** to download the private key, note the **Key ID** shown, and then choose **Done**. You will need this private key and the **Key ID** value shown on this page after you choose Apple as your identity provider in Using social identity providers with a user pool.

Add a social IdP to your user pool

In this section, you configure a social IdP in your user pool using the client ID and client secret from the previous section.

To configure a user pool social identity provider with the AWS Management Console

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Social and external providers** menu. Locate **Federated sign-in** and select **Add an identity provider**.
5. Choose a social identity provider: **Facebook**, **Google**, **Login with Amazon**, or **Sign in with Apple**.
6. Choose from the following steps, based on your choice of social identity provider:

    - **Google** and **Login with Amazon** – Enter the **app client ID** and **app client secret** that was generated in the previous section.
    - **Facebook** – Enter the **app client ID** and **app client secret** that was generated in the previous section, and then choose an API version (for example, version 2.12). We recommend

choosing the latest possible version—each Facebook API has a lifecycle and deprecation date. Facebook scopes and attributes can vary between API versions. We recommend testing your social identity log in with Facebook to ensure that federation works as intended.

- **Sign in with Apple** – Enter the **Services ID**, **Team ID**, **Key ID**, and **private key** that was generated in the previous section.

7. Enter the names of the **Authorized scopes** that you want to use. Scopes define which user attributes (such as `name` and `email`) you want to access with your app. For Facebook, these should be separated by commas. For Google and Login with Amazon, they should be separated by spaces. For Sign in with Apple, select the check boxes for the scopes you want access to.

Social identity provider	Example scopes
Facebook	`public_profile, email`
Google	`profile email openid`
Login with Amazon	`profile postal_code`
Sign in with Apple	`email name`

Your app user is prompted to consent to providing these attributes to your app. For more information about social provider scopes, see the documentation from Google, Facebook, Login with Amazon, or Sign in with Apple.

With Sign in with Apple, the following are user scenarios where scopes might not be returned:

- An end user encounters failures after leaving Apple's sign in page (these can be from internal failures within Amazon Cognito or anything written by the developer).
- The service ID identifier is used across user pools and/or other authentication services.
- A developer adds additional scopes after the user signs in. Users only retrieve new information when they authenticate and when they refresh their tokens.
- A developer deletes the user and then the user signs in again without removing the app from their Apple ID profile.

8. Map attributes from your identity provider to your user pool. For more information, see Things to know about mappings.
9. Choose **Create**.

10. From the **App clients** menu, choose one of the app clients in the list and **Edit hosted UI settings**. Add the new social identity provider to the app client under **Identity providers**.
11. Choose **Save changes**.

Test your social IdP configuration

You can create a login URL by using the elements from the previous two sections. Use it to test your social IdP configuration.

```
https://mydomain.auth.us-east-1.amazoncognito.com/login?
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

You can find your domain on the user pool **Domain name** console page. The client_id is on the **App client settings** page. Use your callback URL for the **redirect_uri** parameter. This is the URL of the page where your user will be redirected after a successful authentication.

> **Note**
>
> Amazon Cognito cancels authentication requests that do not complete within 5 minutes, and redirects the user to managed login. The page displays a `Something went wrong` error message.

Add a SAML 2.0 identity provider

Your app users can sign in with a SAML 2.0 identity provider (IdP). You might choose SAML 2.0 IdPs over social IdPs when your customers are the internal customers or linked businesses of your organization. Where a social IdP permits all users to register for an account, a SAML IdP is more likely to pair with a user directory that your organization controls. Whether your users sign in directly or through a third party, all users have a profile in the user pool. Skip this step if you don't want to add sign in through a SAML identity provider.

For more information, see Using SAML identity providers with a user pool.

You must update your SAML identity provider and configure your user pool. For information about how to add your user pool as a relying party or application for your SAML 2.0 identity provider, see the documentation for your SAML identity provider.

You must also provide an assertion consumer service (ACS) endpoint to your SAML identity provider. Configure the following endpoint in your user pool domain for SAML 2.0 POST binding in your SAML identity provider. For more information about user pool domains, see Configuring a user pool domain.

```
https://Your user pool domain/saml2/idpresponse
With an Amazon Cognito domain:
https://<yourDomainPrefix>.auth.<region>.amazoncognito.com/saml2/idpresponse
With a custom domain:
https://Your custom domain/saml2/idpresponse
```

You can find your domain prefix and the Region value for your user pool in the **Domain** menu in the Amazon Cognito console.

For some SAML identity providers, you also need to provide the service provider (SP) `urn`, also called the audience URI or SP entity ID, in the format:

```
urn:amazon:cognito:sp:<yourUserPoolID>
```

You can find your user pool ID in the **Overview** dashboard for your user pool in the Amazon Cognito console.

You should also configure your SAML identity provider to provide attribute values for any attributes that are required in your user pool. Typically, `email` is a required attribute for user pools. In that case, the SAML identity provider should provide an `email` value (claim) in the SAML assertion.

Amazon Cognito user pools support SAML 2.0 federation with post-binding endpoints. This eliminates the need for your app to retrieve or parse SAML assertion responses because the user pool directly receives the SAML response from your identity provider through a user agent.

To configure a SAML 2.0 identity provider in your user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Social and external providers** menu. Locate **Federated sign-in** and select **Add an identity provider**.
5. Choose a **SAML** social identity provider.

6. Enter **Identifiers** separated by commas. An identifier tells Amazon Cognito it should check the email address that a user enters when they sign in. Then it directs them to the provider that corresponds to their domain.
7. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. You must configure your SAML 2.0 identity provider to send sign-out responses to the `https://<your Amazon Cognito domain>/saml2/logout` endpoint that is created when you configure managed login. The `saml2/logout` endpoint uses the POST binding.

   > **Note**
   >
   > If this option is selected and your SAML identity provider expects a signed logout request, you will also need to configure the signing certificate that is provided by Amazon Cognito with your SAML IdP.
   > The SAML IdP will process the signed logout request and will log out your user from the Amazon Cognito session.

8. Choose a **Metadata document source**. If your identity provider offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

   > **Note**
   >
   > We recommend that you enter a metadata document URL if your provider has a public endpoint, rather than uploading a file. This allows Amazon Cognito to refresh the metadata automatically. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

9. Select **Map attributes between your SAML provider and your app** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

   For example, when you choose the **User pool attribute** `email`, enter the SAML attribute name as it appears in the SAML assertion from your identity provider. Your identity provider might offer sample SAML assertions for reference. Some identity providers use simple names, such as `email`, while others use URL-formatted attribute names, such as the following example:

```
http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
```

10. Choose **Create**.

Guided setup options for Amazon Cognito

You might want to evaluate the features of Amazon Cognito in a structured, guided experience. Here are some external resources that provide tailored experiences with user pools and identity pools.

Complete a workshop

AWS workshop studio hosts a workshop that walks you through the setup of the majority of Amazon Cognito features. These features include the user pools API, the user pools hosted UI, identity pools, and security configuration.

Add application code from examples

The code examples chapter in this guide has application code that you can use with user pools and identity pools. The user pools section of the code examples chapter has short snippets that cover individual operations, and longer examples for end-to-end example applications in a variety of programming languages.

Create a fullstack application with AWS Amplify

AWS Amplify is an AWS service for developers who want to develop and host an application and user interface. Amazon Cognito is the authentication component of Amplify. When you add authentication to your application, Amplify can automate the deployment of Amazon Cognito user pool and identity pool resources. Also, see Integrating Amazon Cognito authentication and authorization with web and mobile apps.

More Amazon Cognito application resources on GitHub

- Authentication flow examples with .NET for Amazon Cognito
- Amazon Cognito Passwordless Auth
- *PetStore* example with Amazon Verified Permissions
- Sample React App Using ABAC + Identity Pools to Access AWS Resources
- Amazon Cognito and API Gateway based machine to machine authorization using AWS CDK

- Building fine-grained authorization using Amazon Cognito, API Gateway, and IAM
- CloudFront authorization@edge

More workshops

- Implement Passwordless authentication with Amazon Cognito and WebAuthn
- Amazon Cognito Workshop
- Amazon Cognito Troubleshooting Workshop
- Authentication and authorization with Amazon Cognito and Verified Permissions
- Amazon Cognito JWT Deep Dive

Blogposts

- Protect public clients for Amazon Cognito by using an Amazon CloudFront proxy
- How to set up Amazon Cognito for federated authentication using Azure AD
- Simplify web app authentication: A guide to AD FS federation with Amazon Cognito user pools

Planning your Amazon Cognito design

The choices in this chapter are easier to make up front than to change later. This chapter covers:

- **Choosing an authentication approach**—how your users sign in.
- **Common scenarios**—architecture patterns to match against your application.
- **Multi-tenancy**—if you serve more than one customer organization, strategies to isolate each tenant's users and data are covered in Building B2B multi-tenant applications.

Choose your authentication approach

When you authenticate users with an Amazon Cognito user pools, you choose how the sign-in experience is presented. The following three approaches differ in how much of the interface Amazon Cognito provides versus how much you build, and they are not mutually exclusive—you can combine them. They all authenticate against the same user pool and produce the same tokens. The difference is where the sign-in pages live and how much control you have.

Managed login (Amazon Cognito hosts the pages)

Amazon Cognito serves the sign-up, sign-in, multi-factor authentication, and password-reset pages from your user pool domain. You customize their appearance with the branding editor. Choose this when you want a complete, hosted sign-in experience without building UI, and when you want the standards-based OAuth 2.0 and OpenID Connect flows handled for you. Managed login is the current version of these Amazon Cognito-hosted pages; the earlier *classic hosted UI* is a simpler, first-generation alternative that you can still select per app client. See the section called “Managed login”.

User pools API (you build the interface)

Your application collects credentials and calls the Amazon Cognito user pools API directly through an AWS SDK. You build and control the entire interface. Choose this when you need a fully custom sign-in experience, or when you are signing in users from a back-end service. See the section called “Authentication”.

Third-party identity providers (federation)

Your users sign in through an external provider. This can be a social provider like Google or Apple, or an enterprise SAML or OIDC provider. Amazon Cognito federates their identity into

your user pool. Federated sign-in runs through the user pool domain's hosted sign-in endpoints—either managed login, or the authorization and login endpoints that you redirect to from a custom interface. The native user pools API operation `InitiateAuth` does not sign a user in through a federated provider, so federation always uses the hosted endpoints rather than the direct SDK authentication flow. Choose this when your users already have identities elsewhere. See the section called “Third-party IdP sign-in”.

The following table summarizes when to reach for each approach.

If you want to	Start with
Ship sign-in fast with no UI to build and maintain	Managed login
Fully control the look and behavior of every sign-in screen	The user pools API
Authenticate requests from a back-end or machine-to-machine caller	The user pools API (and, for M2M, client credentials)
Let users sign in with Google, Apple, or an enterprise directory	Third-party identity providers (federation), through managed login or the hosted authorization and login endpoints

Note

Whichever approach you choose, your user pool's feature plan determines which capabilities are available. The **Lite** plan covers core sign-up and sign-in; **Essentials** adds options such as passwordless and passkey sign-in and managed login; and **Plus** adds advanced threat protection. You select a plan when you create a user pool and can change it later, so you don't have to decide before you start. If a feature you expect isn't available, check your plan first. For a full comparison, see the section called “User pool feature plans”.

> **Note**
>
> These approaches all answer *who is this user?* (authentication). If you also need to grant users temporary AWS credentials to reach services like Amazon S3, that is a separate question—authorization—answered by identity pools. See the section called "Which parts of Amazon Cognito do you need?".

Common Amazon Cognito scenarios

This topic describes seven common scenarios for using Amazon Cognito.

The two main components of Amazon Cognito are user pools and identity pools. User pools are user directories that provide sign-up and sign-in options for your web and mobile app users. Identity pools provide temporary AWS credentials to grant your users access to other AWS services.

A user pool is a user directory in Amazon Cognito. Your app users can either sign in directly through a user pool, or they can federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs. Whether your users sign in directly or through a third party, all members of the user pool have a directory profile that you can access through an SDK.

With an identity pool, your users can obtain temporary AWS credentials to access AWS services, such as Amazon S3 and DynamoDB. Identity pools support anonymous guest users, as well as federation through third-party IdPs.

Topics

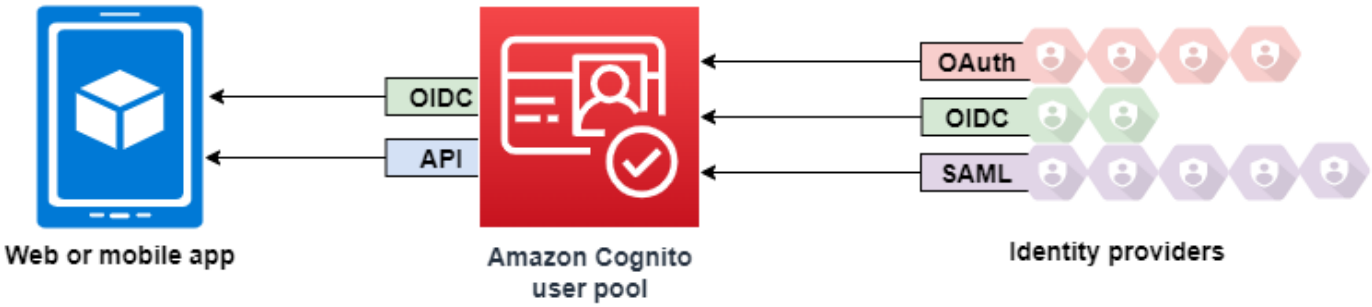
- Authenticate with a user pool
- Access back-end resources with user pool tokens
- Access resources with API Gateway and Lambda with a user pool
- Access AWS services with a user pool and an identity pool
- Authenticate with a third party and access AWS services with an identity pool
- Access AWS AppSync resources with Amazon Cognito
- Authorize machine-to-machine (M2M) applications
- Step-up authentication

Authenticate with a user pool

You can enable your users to authenticate with a user pool. Your app users can either sign in directly through a user pool, or they can federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs.

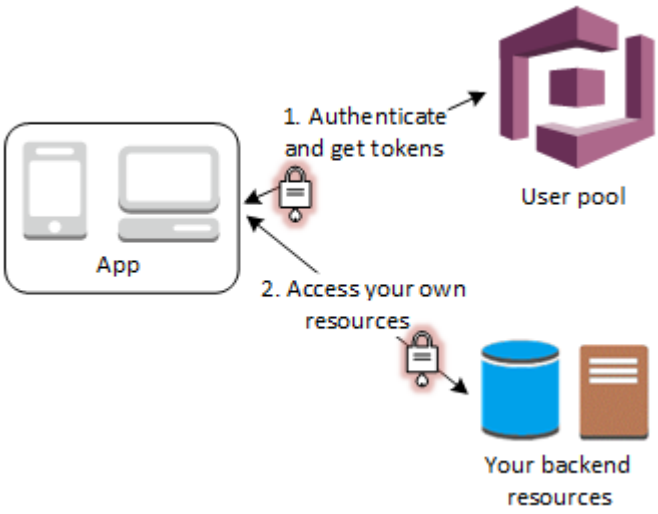
After a successful authentication, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to retrieve AWS credentials that allow your app to access other AWS services, or you might choose to use them to control access to your server-side resources, or to the Amazon API Gateway.

For more information, see An example authentication session and Understanding user pool JSON web tokens (JWTs).



Access back-end resources with user pool tokens

After a successful user pool sign-in, your web or mobile app will receive user pool tokens from Amazon Cognito. You can use those tokens to control access to your server-side resources. You can also create user pool groups to manage permissions, and to represent different types of users. For more information on using groups to control access to your resources, see Adding groups to a user pool.



After you configure a domain for your user pool, Amazon Cognito provisions a hosted web UI that allows you to add sign-up and sign-in pages to your app. Using this OAuth 2.0 foundation, you can create your own resource server to enable your users to access protected resources. For more information, see Scopes, M2M, and resource servers.

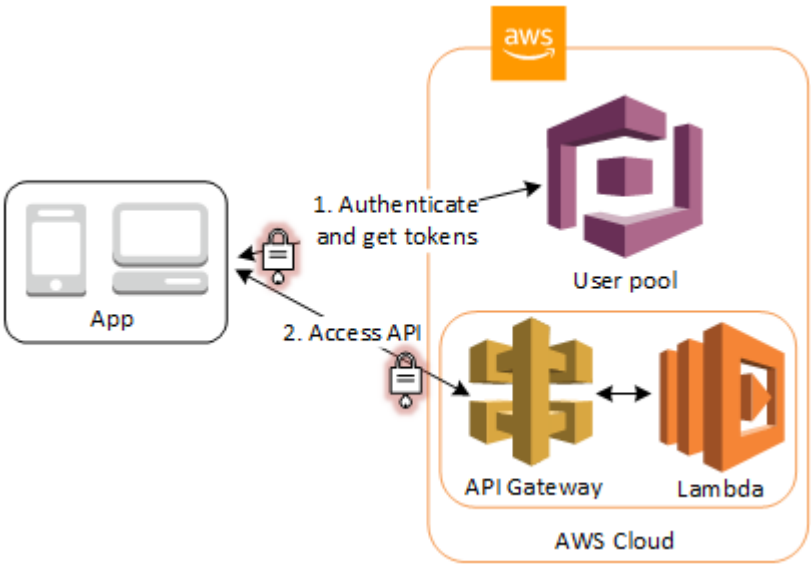
For more information about user pool authentication, see An example authentication session and Understanding user pool JSON web tokens (JWTs).

Access resources with API Gateway and Lambda with a user pool

You can enable your users to access your API through API Gateway. API Gateway validates the tokens from a successful user pool authentication, and uses them to grant your users access to resources including Lambda functions, or your own API.

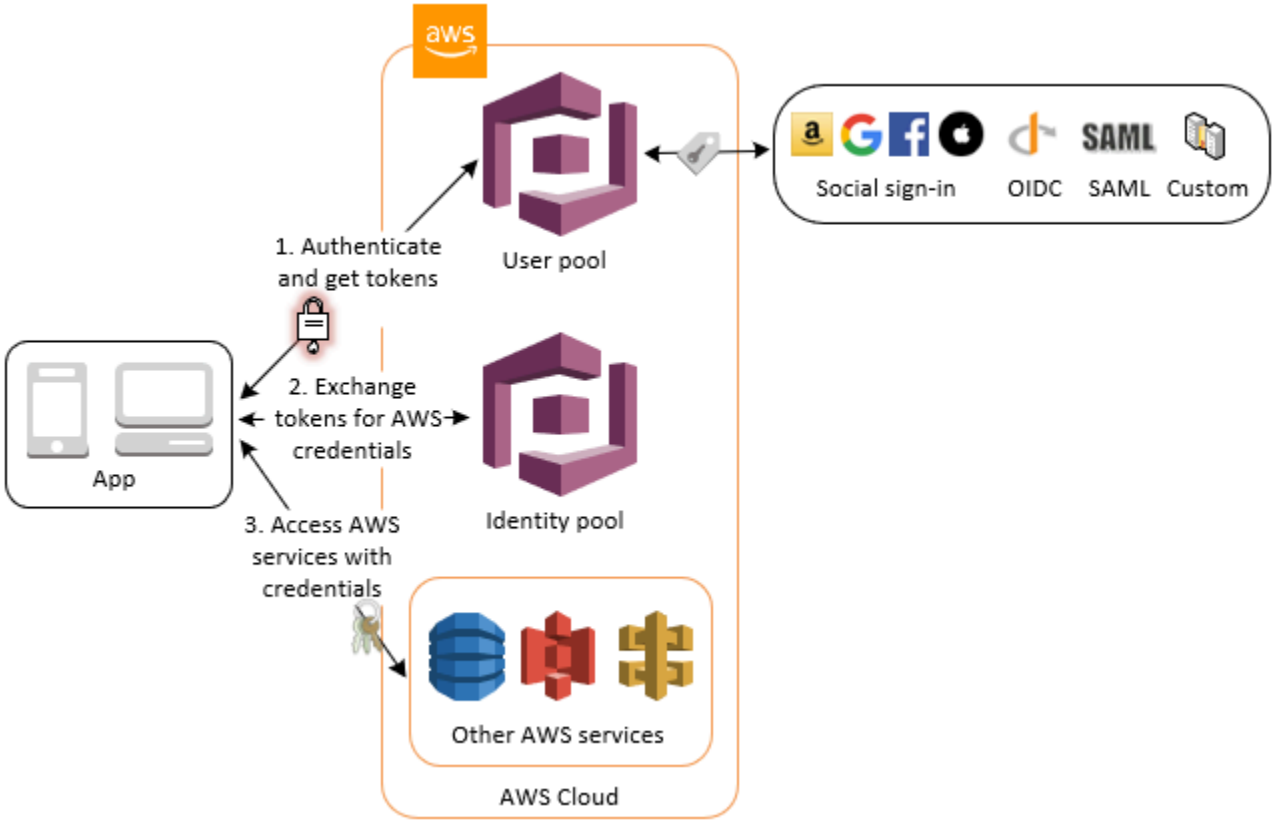
You can use groups in a user pool to control permissions with API Gateway by mapping group membership to IAM roles. The groups that a user is a member of are included in the ID token provided by a user pool when your app user signs in. For more information on user pool groups See Adding groups to a user pool.

You can submit your user pool tokens with a request to API Gateway for verification by an Amazon Cognito authorizer Lambda function. For more information on API Gateway, see Using API Gateway with Amazon Cognito user pools.



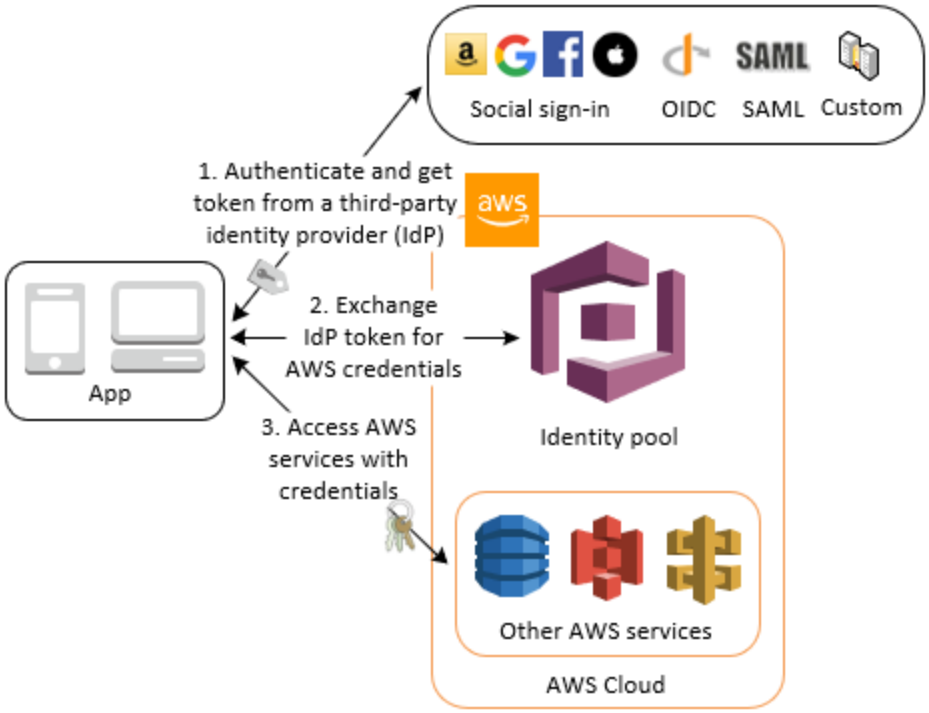
Access AWS services with a user pool and an identity pool

After a successful user pool authentication, your app will receive user pool tokens from Amazon Cognito. You can exchange them for temporary access to other AWS services with an identity pool. For more information, see Accessing AWS services using an identity pool after sign-in and Getting started with Amazon Cognito identity pools.



Authenticate with a third party and access AWS services with an identity pool

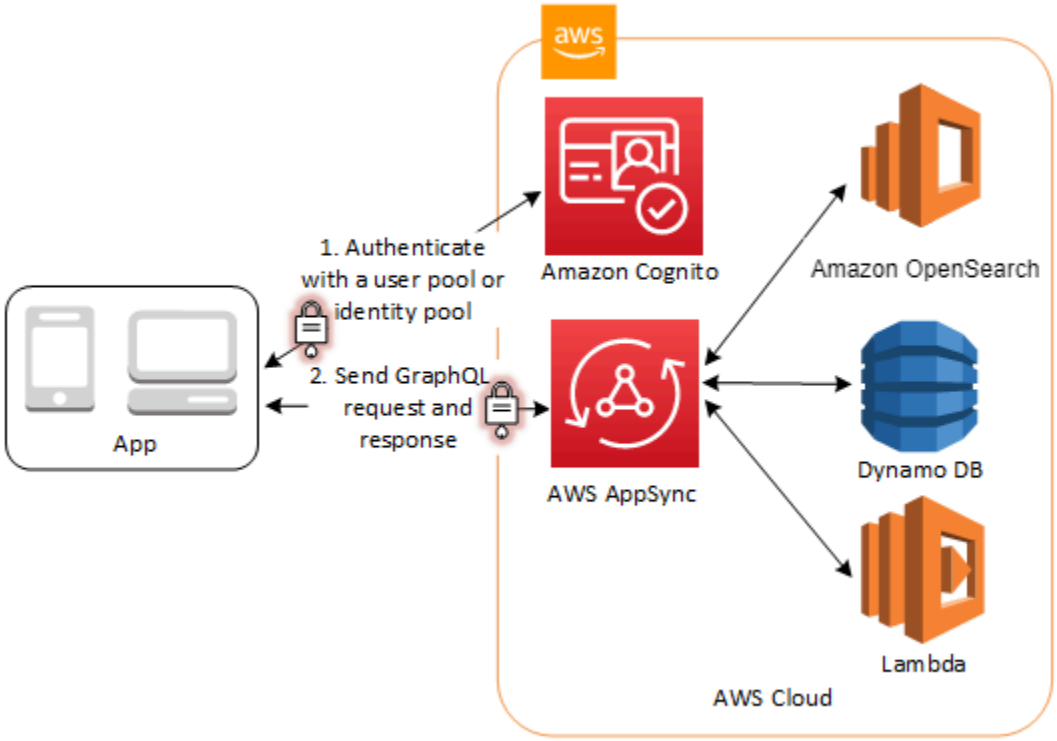
You can enable your users access to AWS services through an identity pool. An identity pool requires an IdP token from a user that's authenticated by a third-party identity provider (or nothing if it's an anonymous guest). In exchange, the identity pool grants temporary AWS credentials that you can use to access other AWS services. For more information, see Getting started with Amazon Cognito identity pools.



Access AWS AppSync resources with Amazon Cognito

You can grant your users access to AWS AppSync resources with tokens from a successful Amazon Cognito user pool authentication. For more information, see AMAZON_COGNITO_USER_POOLS authorization in the *AWS AppSync Developer Guide*.

You can also sign requests to the AWS AppSync GraphQL API with the IAM credentials that you receive from an identity pool. See AWS_IAM authorization.



Authorize machine-to-machine (M2M) applications

With machine-to-machine (M2M) authorization, an application, service, or device can authenticate as itself and get authorized access to your APIs, with no signed-in user. Amazon Cognito authorizes an application instead of a user and issues only an access token—no ID token and no refresh token. This is the model for workloads such as an application that autonomously collects data from a fleet of Internet of Things (IoT) devices, where each device connects to your API and pushes data with no human in the loop.

To enable M2M authorization, build an app client that has a client secret, and enable it for the `client_credentials` grant. Your machine identity requests tokens from the authorization server and presents them to your resource server. For more information, see Machine-to-machine (M2M) authorization and Token endpoint.

Step-up authentication

Consider a banking application that lets users sign in with a username and password. To make a wire transfer, the application requires multi-factor authentication (MFA). When a user attempts a transfer, the application inspects the `acr` claim in the user's access and ID tokens, determines that

the authentication level is insufficient, and redirects the user to Amazon Cognito to complete MFA. Amazon Cognito then issues a new token with an elevated authentication level, and the application allows the transfer.

Before you redirect a user to Amazon Cognito for step-up authentication, evaluate the tokens that your application already holds. Compare the `auth_time` claim to the maximum age that the operation allows, and compare the token's `acr` level to the level that the operation requires. Redirect the user to Amazon Cognito to complete the authentication flow only when the required authentication level or freshness isn't already met.

For more information about ACR levels, AMR values, and other step-up authentication concepts, see the section called “Authentication levels with ACR and AMR claims”. For the mechanics of requesting step-up authentication, see Step-up authentication with the USER_AUTH flow for the USER_AUTH flow, and Authorize endpoint and Login endpoint for managed login.

Setting up your sign-in surface

Before your users can authenticate, you set up the surface they sign in through. For managed login, that means a user pool domain to serve the pages and an app client that points to them, plus the branding you apply. This chapter covers:

- **Managed login**—the hosted sign-in pages, the user pool domain that serves them, and the branding editor.
- **Migrating from the classic hosted UI**—what changes when you move an existing user pool to managed login.

To decide between managed login, the user pools API, and federation in the first place, see the section called "Choose an approach". For the authentication flows themselves, see the section called "Authentication".

User pool managed login

You can choose a web domain to host services for your user pool. An Amazon Cognito user pool gains the following functions when you add a domain, collectively referred to as *managed login*.

- An authorization server that acts as an identity provider (IdP) to applications that work with OAuth 2.0 and OpenID Connect (OIDC). The authorization server routes requests, issues and manages JSON web tokens (JWTs), and delivers user attribute information.
- A ready-to-use user interface (UI) for authentication operations like sign-in, sign-out and password management. The *managed login pages* act as a web front end for authentication services.
- A service provider (SP), or relying party (RP), to SAML 2.0 IdPs, OIDC IdPs, Facebook, Login with Amazon, Sign in with Apple, and Google.

Managed login has an alternative: the classic *hosted UI*. You choose one or the other for a user pool domain—managed login and the classic hosted UI aren't used together. The classic hosted UI is a first-generation version of the managed login services. Hosted UI IdP and RP services generally have the same characteristics as managed login, but the login pages have a simpler design and fewer features. For example, passkey sign-in isn't available in the classic hosted UI. In the Lite feature plan, the classic hosted UI is your only option for user pool domain services.

The managed login pages are a collection of web interfaces for basic sign-up, sign-in, multi-factor authentication and password-reset activities in your user pool. They also connect users to one or more third-party identity providers (IdPs) when you want to give users a choice of sign-in option. Your app can invoke your managed login pages in users' browsers when you want to authenticate and authorize users.

You can make the managed login user experience fit your brand with custom logos, backgrounds and styles. You have two options for the branding that you might apply to your managed login UI: the *branding editor* for managed login, and *hosted UI (classic) branding* for the hosted UI.

Branding editor

An updated user experience with the most up-to-date authentication options and a visual editor in the Amazon Cognito console.

Hosted UI (classic) branding

A familiar user experience for previous adopters of Amazon Cognito user pools. Branding for the hosted UI is a file-based system. To apply branding to hosted UI pages, you upload a logo image file and a file that sets values for several predetermined CSS style options.

The branding editor isn't available in all feature plans for user pools. For more information, see User pool feature plans.

For more information about constructing requests to managed login and hosted UI services, see User pool endpoints and managed login reference.

Managed login supports step-up authentication with the `acr_values` and `max_age` parameters on the authorize and login endpoints. For more information, see the authorize endpoint and login endpoint.

> **Note**
>
> Amazon Cognito managed login doesn't support custom authentication with custom authentication challenge Lambda triggers.

Topics

- Managed login localization

- Terms documents
- Setting up managed login with the Amazon Cognito console
- Setting up managed login with AWS Amplify
- Viewing your sign-in page
- Customizing your authentication pages
- Things to know about managed login and the hosted UI
- Configuring a user pool domain
- Apply branding to managed login pages

Managed login localization

Managed login defaults to the English language in user-interactive pages. You can display your managed login pages localized for the language of your choice. The available languages are those available in the AWS Management Console. In the link that you distribute to users, add a `lang` query parameter, as shown in the following example.

```
https://<your domain>/oauth2/authorize?lang=es&response_type=code&client_id=<your app
 client id>&redirect_uri=<your relying-party url>
```

Amazon Cognito sets a cookie in users' browser with their language preference after the initial request with a `lang` parameter. After the cookie is set, the language selection persists without displaying or requiring you to include the parameter in requests. For example, after a user makes a sign-in request with a `lang=de` parameter, their managed login pages render in German until they clear their cookies or make a new request with a new localization parameter like `lang=en`.

Localization is only available for managed login. You must be on the Essentials or Plus feature plan and have assigned your domain to use managed login branding.

The selection that your user makes in managed login isn't available to custom email or SMS sender triggers. When you implement these triggers, you must use other mechanisms to determine a user's preferred language. In sign-in flows, the `locale` attribute might indicate the user's preferred language based on location. In sign-up flows, the region or app client ID of your user pool might indicate a language preference.

The following languages are available.

Managed login languages

Language	Code
German	de
English	en
Spanish	es
French	fr
Bahasa Indonesia	id
Dutch	nl
Italian	it
Japanese	ja
Korean	ko
Portuguese (Brazil)	pt-BR
Chinese (Simplified)	zh-CN
Chinese (Traditional)	zh-TW

Terms documents

You can configure your managed login pages to display links to your **Terms of use** and **Privacy policy** documents when users sign up. When you set up both terms documents in your app client, users see the following text during sign-up: **By signing up, you agree to our Terms of use and Privacy policy.** The phrases **Terms of use** and **Privacy policy** appear in your managed login page, hyperlinked to your documents.

Terms documents support language-specific URLs that align with managed login localization. When users select a language with the `lang` query parameter, Amazon Cognito displays links to your terms documents in that language. If you haven't configured a URL for a specific language, Amazon Cognito uses the default URL that you configured for the app client.

To configure terms documents for your app client, navigate to the **Managed login** menu in your user pool. Under **Terms documents**, choose **Create terms document**.

Amazon Cognito console

To create a terms document

1. Navigate to your user pool and choose the **Managed login** menu. Locate **Terms documents**.
2. Choose **Create terms document**.
3. Select the app client that you want to assign the terms document to.
4. Enter a **Terms name**. This identifies your document in the console.
5. Under **Links**, choose a **Language** and enter the **URL** where you host your terms document in that language.
6. To add URLs for additional languages, choose **Add another**.
7. Choose **Create**.

Amazon Cognito user pools API

The following is an example CreateTerms request body. It causes the sign-up page for app client `1example23456789` to render links for a french-language and a portuguese-language (Brazil) version of the privacy policy when managed login is localized to that language. A separate request is required to set URLs for `terms-of-use` before managed login will render links at the sign-up page.

```
{
   "ClientId": "1example23456789",
   "Enforcement": "NONE",
   "Links": {
      "cognito:default" : "https://example.com/privacy/",
      "cognito:french" : "https://example.com/fr/privacy/",
      "cognito:portuguese-brazil" : "https://example.com/pt/privacy/"
   },
   "TermsName": "privacy-policy",
   "TermsSource": "LINK",
   "UserPoolId": "eusc-de-east-1_EXAMPLE"
}
```

> **Note**
>
> You must create both a terms of use and a privacy policy document for your app client before Amazon Cognito will display terms documents in your managed login pages.

Setting up managed login with the Amazon Cognito console

The first requirement for managed login and hosted UI is a user pool domain. In the user pools console, navigate to the **Domain** tab of your user pool and add a **Cognito domain** or a **custom domain**. You can also choose a domain during the process of creating a new user pool. For more information, see Configuring a user pool domain. When a domain is active in your user pool, all app clients serve public authentication pages on that domain.

When you create or modify a user pool domain, you set the **Branding version** for your domain. This branding version is a choice of **Managed login** or **Hosted UI (classic)**. Your choice of branding version applies to all app clients that use the sign-in services at your domain.

The next step is to create an app client from the **App clients** tab of your user pool. In the process of creating an app client, Amazon Cognito will ask you for information about your application, then prompt you to select a **Return URL**. The return URL is also called the relying party (RP) URL, the redirect URI, and the callback URL. This is the URL that your application runs from, for example `https://www.example.com` or `myapp://example`.

The branding version that you set on the domain determines the *style* of your sign-in pages: choose **Managed login** to design pages in the branding editor, or **Hosted UI (classic)** to apply file-based hosted UI branding. For more information about the branding editor and the feature plans that support it, see User pool feature plans.

After you configure a domain and an app client and set a branding version in your user pool, your managed login pages become available on the internet.

To set up managed login in the Amazon Cognito console

1. Add a user pool domain. In the Amazon Cognito console, choose your user pool, open the **Domain** tab, and add a **Cognito domain** (prefix domain) or a **custom domain**. For more information, see Configuring a user pool domain.

2. Set the branding version for the domain. When you create or modify the domain, choose **Managed login** or **Hosted UI (classic)**. This choice sets the style of your sign-in pages and applies to all app clients that use the domain.
3. Create an app client. On the **App clients** tab, create an app client and set its return URL (also called the callback URL or redirect URI) to the URL that your application runs from.
4. (Managed login only) Customize the style. Open the branding editor for your app client to design your sign-in pages. The branding editor is available in a subset of feature plans; for more information, see User pool feature plans.
5. View your sign-in page. On the **Login pages** tab for your app client, choose **View login pages** to open your managed login pages.

Setting up managed login with AWS Amplify

If you use AWS Amplify to add authentication to your web or mobile app, you can set up your managed login pages in the Amplify command line interface (CLI) and libraries in the Amplify framework. To add authentication to your app, add the `Auth` category to your project. Then, in your application, authenticate user pool users with Amplify client libraries.

You can invoke managed login pages for authentication or you can federate users through an authorization endpoint that redirects to an IdP. After a user successfully authenticates with the provider, Amplify creates a new user in your user pool and passes the user's tokens to your app.

The following examples show how to use AWS Amplify to set up managed login with social providers in your app.

- React
- Swift
- Flutter
- Android

Viewing your sign-in page

In the Amazon Cognito console, choose the **View login pages** button in the **Login pages** tab for your app client under the **App clients** menu. This button takes you to a sign-in page in your user pool domain with the following basic parameters.

- The app client id
- An authorization code grant request
- A request for all scopes that you have activated for the current app client
- The first callback URL in the list for the current app client

The **View login page** button is useful when you want to test the basic functions of your managed login pages. Your login pages will match the **Branding version** that you assigned to your user pool domain. You can customize your sign-in URL with additional and modified parameters. In most cases, the automatically-generated parameters of the **View login page** link don't fully match the needs of your app. In these cases, you must customize the URL that your app invokes when it signs in your users. For more information about sign-in parameter keys and values, see User pool endpoints and managed login reference.

The sign-in webpage uses the following URL format. This example requests an authorization code grant with the `response_type=code` parameter.

```
https://<your domain>/oauth2/authorize?response_type=code&client_id=<your app client
 id>&redirect_uri=<your relying-party url>
```

You can look up your user pool domain string from the **Domain** menu in your user pool. In the **App clients** menu, you can identify app client IDs, their callback URLs, their allowed scopes, and other configuration.

When you navigate to the `/oauth2/authorize` endpoint with your custom parameters, Amazon Cognito either redirects you to the `/oauth2/login` endpoint or, if you have an `identity_provider` or `idp_identifier` parameter, silently redirects you to your IdP sign-in page.

Example request for an implicit grant

You can view your sign-in webpage with the following URL for the implicit code grant where `response_type=token`. After a successful sign-in, Amazon Cognito returns user pool tokens to your web browser's address bar.

```
            https://mydomain.auth.us-east-1.amazoncognito.com/authorize?
response_type=token&client_id=1example23456789&redirect_uri=https://
mydomain.example.com
```

The identity and access tokens appear as parameters appended to your redirect URL.

The following is an example response from an implicit grant request.

```
            https://auth.example.com/
#id_token=eyJraaBcDeF1234567890&access_token=eyJraGhIjKlM1112131415&expires_in=3600&token_type=
```

Customizing your authentication pages

In the past, Amazon Cognito only hosted login pages with the classic *hosted UI*, a simple design that grants a universal look to authentication webpages. You could customize Amazon Cognito user pools with a logo image and tweak some styles with a file that specified some CSS style values. Later, Amazon Cognito introduced *managed login*, an updated hosted authentication service. Managed login is an updated look-and-feel with the *branding editor*. The branding editor is a no-code visual editor and a larger suite of options than the hosted UI customization experience. Managed login also introduced custom background images and a dark mode theme.

You can switch between the managed login and hosted UI branding experiences in user pools. To learn more about customizing your managed login pages, see Apply branding to managed login pages.

Things to know about managed login and the hosted UI

The one-hour managed login and hosted UI session cookie

When a user signs in with your login pages or a third-party provider, Amazon Cognito sets a cookie in their browser. With this cookie, users can sign in again with the same authentication method for one hour. When they sign in with their browser cookie, they get fresh tokens that last the duration specified in your app client configuration. Changes to user attributes or authentication factors have no effect on their ability to sign in again with their browser cookie.

Authentication with the session cookie doesn't reset the cookie duration to an additional hour. Users must sign in again if they attempt to access your sign-in pages more than an hour after their most recent successful interactive authentication.

Confirming user accounts and verifying user attributes

For user pool local users, managed login and the hosted UI work best when you configure your user pool to **Allow Cognito to automatically send messages to verify and confirm**. When you enable this setting, Amazon Cognito sends a message with a confirmation code to users who sign up. When you instead confirm users as a user pool administrator, your login pages display an error message after sign-up. In this state, Amazon Cognito has created the new user, but hasn't been able to send a verification message. You can still confirm users as an administrator, but they might contact your support desk after they encounter an error. For more information about administrative confirmation, see Allowing users to sign up in your app but confirming them as a user pool administrator.

Managed login scope of operations

Managed login and the classic hosted UI support sign-up, sign-in, and password management. This includes completing sign-in with multi-factor authentication (MFA) and registration of webAuthn authenticators. Managed login doesn't support user self-service profile management like attribute changes and setting of MFA preference. You must implement profile management in your own application code. Managed login also doesn't provide the capacity to confirm attribute changes when you update email addresses and phone numbers as an administrator with the AdminUpdateUserAttributes API operation.

Viewing your changes to configuration

If you make style changes to your pages and they do not immediately appear, wait a few minutes and then refresh the page.

Decoding user pool tokens

Amazon Cognito user pool tokens are signed using an RS256 algorithm. You can decode and verify user pool tokens using AWS Lambda. See Decode and verify Amazon Cognito JWT tokens on GitHub.

TLS version

Managed login and hosted UI pages require encryption in transit. User pool domains that are provided by Amazon Cognito require that users' browsers negotiate a minimum TLS version of 1.2. Custom domains support browser connections with TLS version 1.2. The hosted UI (classic) **doesn't require** TLS 1.2 for custom domains, but the newer managed login **requires** TLS version 1.2 both for custom and prefix domains. Because Amazon Cognito manages the configuration of your domain services, you can't modify the TLS requirements of your user pool domain.

CORS policies

Neither managed login nor the hosted UI support custom cross-origin resource sharing (CORS) origin policies. A CORS policy would prevent users from passing authentication parameters in their requests. Instead, implement a CORS policy in your application front end. Amazon Cognito returns an `Access-Control-Allow-Origin: *` response header to requests to the following endpoints.

1. Token endpoint
2. Revoke endpoint
3. userInfo endpoint

Cookies

Managed login and the hosted UI set cookies in users' browsers. The cookies conform to the requirements of some browsers that sites not set third-party cookies. They are scoped only to your user pool endpoints and include the following:

- An `XSRF-TOKEN` cookie for each request.
- A `csrf-state` cookie for session consistency when a user is redirected.
- A `csrf-state-legacy` cookie for session consistency, read by Amazon Cognito as a fallback when your browser doesn't have support for the `SameSite` attribute.
- A `cognito` session cookie that preserves successful sign-in attempts for an hour.
- A `lang` cookie that preserves a user's choice of language localization in managed login.
- A `page-data` cookie that persists required data as a user navigates between managed login pages.

In iOS, you can block all cookies. This setting isn't compatible with managed login or the hosted UI. To work with users who might enable this setting, build user pool authentication into a native iOS app with an AWS SDK. In this scenario, you can build your own session storage that isn't cookie-based.

Effects of managed login version change

Consider the following effects of adding domains and setting the managed login version.

- When you add a prefix domain, either with managed login or hosted UI (classic) branding, it can take up to 60 seconds before your login pages are available.
- When you add a custom domain, either with managed login or hosted UI (classic) branding, it can take up to five minutes before your login pages are available.

- When you change the branding version for your domain, it can take up to four minutes before your login pages are available in the new branding version.
- When you switch between managed login and hosted UI (classic) branding, Amazon Cognito doesn't maintain user sessions. They must sign in again with the new interface.

Default style

When you create an app client in the AWS Management Console, Amazon Cognito automatically assigns a branding style to your app client. When you programmatically create an app client with the CreateUserPoolClient operation, no branding style is created. Managed login isn't available for an app client created with an AWS SDK until you create one with a CreateManagedLoginBranding request.

Managed login authentication prompt times out

Amazon Cognito cancels authentication requests that do not complete within five minutes, and redirects the user to managed login. The page displays a `Something went wrong` error message.

Configuring a user pool domain

Configuring a domain is an optional part of setting up a user pool. A user pool domain hosts features for user authentication, federation with third-party providers, and OpenID Connect (OIDC) flows. It has *managed login*, a prebuilt interface for key operations like signing up, signing in, and password recovery. It also hosts the standard OpenID Connect (OIDC) endpoints like authorize, userInfo, and token, for machine-to-machine (M2M) authorization and other OIDC and OAuth 2.0 authentication and authorization flows.

M2M authorization is also available without a domain. You can request client-credentials access tokens with the GetClientToken API operation, which uses the Amazon Cognito API instead of the token endpoint.

Users authenticate with managed login pages at the domain associated with your user pool. You have two options for configuring this domain: you can either use the default Amazon Cognito prefix domain, or you can configure a custom domain that you own.

The custom domain option has more options for flexibility, security and control. For example, a familiar, organization-owned domain can encourage user trust and make the sign-in process more intuitive. However, the custom domain approach requires some additional overhead, like managing the SSL certificate and DNS configuration.

The OIDC discovery endpoints, `/.well-known/openid-configuration` for endpoint URLs and `/.well-known/jwks.json` for token signing keys, aren't hosted on your domain. For more information, see Identity provider and relying party endpoints.

Understanding how to configure and manage the domain for your user pool is an important step in integrating authentication into your application. Sign-in with the user pools API and an AWS SDK can be an alternative to configuring a domain. The API-based model delivers tokens directly in an API response, but for implementations that use the extended capabilities of user pools as an OIDC IdP, you must configure a domain. For more information about the authentication models that are available in user pools, see Understanding API, OIDC, and managed login pages authentication.

Topics

- Things to know about user pool domains
- Using the Amazon Cognito prefix domain for managed login
- Using your own domain for managed login

Things to know about user pool domains

User pool domains are a point of service for OIDC relying parties in your applications and for UI elements. Consider the following details when you're planning your implementation of a domain for your user pool.

Reserved terms

You can't use the text `aws`, `amazon`, or `cognito` in the name of an Amazon Cognito prefix domain.

Discovery endpoints are on a different domain

The user pool discovery endpoints `.well-known/openid-configuration` and `.well-known/jwks.json` aren't on your user pool custom or prefix domain. The path to these endpoints is as follows.

- `https://cognito-idp.`*`Region`*`.amazonaws.com/`*`your user pool ID`*`/.well-known/openid-configuration`
- `https://cognito-idp.`*`Region`*`.amazonaws.com/`*`your user pool ID`*`/.well-known/jwks.json`

Effective time of domain changes

It can take Amazon Cognito up to a minute to launch or update the branding version of a prefix domain. Changes to a custom domain can take up to five minutes to propagate. New custom domains can take up to one hour to propagate.

Custom and prefix domains at the same time

You can set up a user pool with both a custom domain and a prefix domain that's owned by AWS. Because the user pool discovery endpoints are hosted at a different domain, they only serve the *custom domain*. For example, your `openid-configuration` will provide a single value for `"authorization_endpoint"` of `"https://auth.example.com/oauth2/authorize"`.

When you have both custom and prefix domains in a user pool, you can use the custom domain with the full features of an OIDC provider. The prefix domain in a user pool with this configuration doesn't have discovery or token-signing-key endpoints and should be used accordingly.

Custom domains preferred as relying party ID for passkey

When you set up user pool authentication with passkeys, you must set a relying party (RP) ID. When you have a custom domain and a prefix domain, you can set the RP ID only as your custom domain. To set a prefix domain as the RP ID in the Amazon Cognito console, delete your custom domain or enter the fully-qualified domain name (FQDN) of the prefix domain as a **Third-party domain**.

Don't use custom domains at different levels of your domain hierarchy

You can configure separate user pools to have custom domains in the same top-level domain (TLD), for example *auth.example.com* and *auth2.example.com*. The managed login session cookie is valid for a custom domain and all subdomains, for example **.auth.example.com*. Because of this, no user of your applications should access managed login for any parent domain *and* subdomain. Where custom domains use the same TLD, keep them at the same subdomain level.

Say you have a user pool with the custom domain *auth.example.com*. Then you create another user pool and assign the custom domain *uk.auth.example.com.*. A user signs in with *auth.example.com.* and gets a cookie that their browser presents to any website in the wildcard path **.auth.example.com*. They then try to sign in to *uk.auth.example.com.*. They pass an invalid cookie to your user pool domain and receive an error instead of a sign-in prompt. By contrast, a user with a cookie for **.auth.example.com* has no issues starting a sign-in session at *auth2.example.com*.

Branding version

When you create a domain, you set a **Branding version**. Your options are the newer managed-login experience and the classic hosted UI experience. This choice applies to all app clients that host services at your domain.

Using the Amazon Cognito prefix domain for managed login

The default experience for managed login is hosted on a domain that AWS owns. This approach has a low barrier to entry—choose a prefix name and it's active—but doesn't have the trust-inspiring features of a custom domain. There isn't a cost difference between the Amazon Cognito domain option and the custom domain option. The only difference is the domain in the web address that you direct your users to. For cases of third-party IdP redirects and client-credentials flows, the prefix domain has little visible effect. A custom domain is better for cases where your users sign in with managed login and would interact with a authentication domain that doesn't match the application domain.

The prefix domain has a prefix of your choosing and is a subdomain of the regional `amazoncognito.com` authentication domain. The following is an example:

```
https://cognitoexample.auth.ap-south-1.amazoncognito.com
```

All prefix domains follow this format: *`prefix`*.auth.*`AWS Region code`*.amazoncognito.com. Custom domain user pools can host the managed login or hosted UI pages on any domain that you own.

> **Note**
>
> To augment the security of your Amazon Cognito applications, the parent domains of user pool endpoints are registered in the Public Suffix List (PSL). The PSL helps your users' web browsers establish a consistent understanding of your user pool endpoints and the cookies they set.
> User pool parent domains take the following formats.
>
> ```
> auth.Region.amazoncognito.com
> auth-fips.Region.amazoncognito.com
> ```
>
> The `auth-fips` domain is a Federal Information Processing Standard (FIPS) 140-3 compliant endpoint. It provides encryption in transit with FIPS-validated cryptographic

modules. Direct your users to this endpoint when your application must meet FIPS compliance requirements. FIPS endpoints for Amazon Cognito managed login are available in a subset of AWS Regions in the US and are not offered in all Regions. For more information about FIPS and the Regions where Amazon Cognito provides FIPS endpoints, see FIPS 140-3.

To add an app client and a user pool domain with the AWS Management Console, see Creating an app client.

Topics

- Prerequisites
- Configure an Amazon Cognito domain prefix
- Verify your sign-in page

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see Create a new application in the Amazon Cognito console.

Configure an Amazon Cognito domain prefix

You can use either the AWS Management Console or the AWS CLI or API to configure a user pool domain.

Amazon Cognito console

Configure a domain

1. Navigate to the **Domain** menu under **Branding**.
2. Next to **Domain**, choose **Actions** and select **Create Cognito domain**. If you have already configured a user pool prefix domain, choose **Delete Cognito domain** before creating your new custom domain.
3. Enter an available domain prefix to use with a **Amazon Cognito domain**. For information on setting up a **Custom domain**, see Using your own domain for managed login.

4. Choose a **Branding version**. Your branding version applies to all user-interactive pages at that domain. Your user pool can host either managed login or hosted UI branding for all app clients.

   > **Note**
   >
   > You can have a custom domain and a prefix domain, but Amazon Cognito only serves the `/.well-known/openid-configuration` endpoint for the *custom* domain.

5. Choose **Create**.

CLI/API

Use the following commands to create a domain prefix and assign it to your user pool.

To configure a user pool domain

- AWS CLI: `aws cognito-idp create-user-pool-domain`

  Example: `aws cognito-idp create-user-pool-domain --user-pool-id <user_pool_id> --domain <domain_name> --managed-login-version 2`
- User pools API operation: CreateUserPoolDomain

To get information about a domain

- AWS CLI: `aws cognito-idp describe-user-pool-domain`

  Example: `aws cognito-idp describe-user-pool-domain --domain <domain_name>`
- User pools API operation: DescribeUserPoolDomain

To delete a domain

- AWS CLI: `aws cognito-idp delete-user-pool-domain`

  Example: `aws cognito-idp delete-user-pool-domain --domain <domain_name>`
- User pools API operation: DeleteUserPoolDomain

Verify your sign-in page

- Verify that the sign-in page is available from your Amazon Cognito hosted domain.

```
https://<your_domain>/login?
response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>
```

Your domain is shown on the **Domain name** page of the Amazon Cognito console. Your app client ID and callback URL are shown on the **App client settings** page.

Using your own domain for managed login

After you set up an app client, you can configure your user pool with a custom domain for the domain services of managed login. With a custom domain, users can sign in to your application using your own web address instead the default `amazoncognito.com` prefix domain. Custom domains improve user trust in your application with a familiar domain name, especially when the root domain matches the domain that hosts your application. Custom domains can improve compliance with organizational security requirements.

A custom domain has some prerequisites, including a user pool, an app client, and a web domain that you own. Custom domains also require an SSL certificate for the custom domain, managed with AWS Certificate Manager (ACM) in US East (N. Virginia). Amazon Cognito creates a Amazon CloudFront distribution, secured in transit with your ACM certificate. Because you own the domain, you must create a DNS record that directs traffic to the CloudFront distribution for your custom domain.

After these elements are ready, you can add the custom domain to your user pool through the Amazon Cognito console or API. This involves specifying the domain name and SSL certificate, and then updating your DNS configuration with the provided alias target. After making these changes, you can verify that the sign-in page is accessible at your custom domain.

The lowest-effort way to create a custom domain is with a public hosted zone in Amazon Route 53. The Amazon Cognito console can create the right DNS records in a few steps. Before you begin, consider creating a Route 53 hosted zone for a domain or subdomain that you own.

Topics

- Adding a custom domain to a user pool

- Prerequisites
- Step 1: Enter your custom domain name
- Step 2: Add an alias target and subdomain
- Step 3: Verify your sign-in page
- Changing the SSL certificate for your custom domain
- Setting the minimum TLS version for your custom domain

Adding a custom domain to a user pool

To add a custom domain to your user pool, you specify the domain name in the Amazon Cognito console, and you provide a certificate you manage with AWS Certificate Manager (ACM). After you add your domain, Amazon Cognito provides an alias target, which you add to your DNS configuration.

Prerequisites

Before you begin, you need:

- A user pool with an app client. For more information, see Create a new application in the Amazon Cognito console.
- A web domain that you own. Its *parent domain* must have a valid DNS **A record**. You can assign any value to this record. The parent may be the root of the domain, or a child domain that is one step up in the domain hierarchy. For example, if your custom domain is *auth.xyz.example.com*, Amazon Cognito must be able to resolve *xyz.example.com* to an IP address. To prevent accidental impact on customer infrastructure, Amazon Cognito doesn't support the use of top-level domains (TLDs) for custom domains. For more information see Domain Names.
- The ability to create a subdomain for your custom domain. We recommend **auth** for your subdomain name. For example: `auth.example.com`.

  > **Note**
  >
  > You might need to obtain a new certificate for your custom domain's subdomain if you don't have a wildcard certificate.

- A public SSL/TLS certificate managed by ACM in US East (N. Virginia). The certificate must be in us-east-1 because the certificate will be associated with a distribution in CloudFront, a global service.

- Browser clients that support Server Name Indication (SNI). The CloudFront distribution that Amazon Cognito assigns to custom domains requires SNI. You can't change this setting. For more information about SNI in CloudFront distributions, see Use SNI to serve HTTPS requests in the *Amazon CloudFront Developer Guide*.
- An application that permits your user pool authorization server to add cookies to user sessions. Amazon Cognito sets several required cookies for managed login pages. These include `cognito`, `cognito-fl`, and `XSRF-TOKEN`. Although each individual cookie conforms to browser size limits, changes to your user pool configuration might cause managed login cookies to grow in size. An intermediate service like an Application Load Balancer (ALB) in front of your custom domain might enforce a maximum header size or total cookie size. If your application also sets its own cookies, your users' sessions might exceed these limits. We recommend that, to avoid size limit conflicts, your application not set cookies on the subdomain that hosts your user pool domain services.
- Permission to update Amazon CloudFront distributions. You can do so by attaching the following IAM policy statement to a user in your AWS account:

  JSON

  ```
  {
      "Version":"2012-10-17",
      "Statement": [
           {
              "Sid": "AllowCloudFrontUpdateDistribution",
              "Effect": "Allow",
              "Action": [
                  "cloudfront:updateDistribution"
              ],
              "Resource": [
                  "*"
              ]
          }
      ]
  }
  ```

  For more information about authorizing actions in CloudFront, see Using Identity-Based Policies (IAM Policies) for CloudFront.

  Amazon Cognito initially uses your IAM permissions to configure the CloudFront distribution, but the distribution is managed by AWS. You can't change the configuration of the CloudFront

distribution that Amazon Cognito associated with your user pool. You can select the minimum TLS version for the distribution security policy when you create or update your custom domain with the Amazon Cognito API. For more information, see Setting the minimum TLS version for your custom domain.

Step 1: Enter your custom domain name

You can add your domain to your user pool by using the Amazon Cognito console or API.

Amazon Cognito console

To add your domain to your user pool from the Amazon Cognito console:

1. Navigate to the **Domain** menu under **Branding**.
2. Next to **Domain**, choose **Actions** and select **Create custom domain** or **Create Amazon Cognito domain**. If you have already configured a user pool custom domain, choose **Delete custom domain** before creating your new custom domain.
3. Next to **Domain**, choose **Actions** and select **Create custom domain**. If you have already configured a custom domain, choose **Delete custom domain** to delete the existing domain before creating your new custom domain.
4. For **Custom domain**, enter the URL of the domain you want to use with Amazon Cognito. Your domain name can include only lowercase letters, numbers, and hyphens. Do not use a hyphen for the first or last character. Use periods to separate subdomain names.
5. For **ACM certificate**, choose the SSL certificate that you want to use for your domain. Only ACM certificates in US East (N. Virginia) are eligible to use with an Amazon Cognito custom domain, regardless of the AWS Region of your user pool.

   If you don't have an available certificate, you can use ACM to provision one in US East (N. Virginia). For more information, see Getting Started in the *AWS Certificate Manager User Guide*.
6. Choose a **Branding version**. Your branding version applies to all user-interactive pages at that domain. Your user pool can host either managed login or hosted UI branding for all app clients.

> **Note**
>
> You can have a custom domain and a prefix domain, but Amazon Cognito only serves the `/.well-known/openid-configuration` endpoint for the *custom* domain.

7. Choose **Create**.
8. Amazon Cognito returns you to the **Domain** menu. A message titled **Create an alias record in your domain's DNS** is displayed. Note down the **Domain** and **Alias target** displayed in the console. They will be used in the next step to direct traffic to your custom domain.

API

The following CreateUserPoolDomain request body creates a custom domain.

```
{
   "Domain": "auth.example.com",
   "UserPoolId": "us-east-1_EXAMPLE",
   "ManagedLoginVersion": 2,
   "CustomDomainConfig": {
    "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "SecurityPolicy": "TLS_V1_3_2025"
   }
}
```

The optional `SecurityPolicy` field sets the minimum TLS version for your custom domain. For more information, see Setting the minimum TLS version for your custom domain.

Step 2: Add an alias target and subdomain

In this step, you set up an alias through your Domain Name Server (DNS) service provider that points back to the alias target from the previous step. If you are using Amazon Route 53 for DNS address resolution, choose the section **To add an alias target and subdomain using Route 53.**

To add an alias target and subdomain to your current DNS configuration

- If you aren't using Route 53 for DNS address resolution, then you must use your DNS service provider's configuration tools to add the alias target from the previous step to your domain's

DNS record. Your DNS provider will also need to set up the subdomain for your custom domain.

To add an alias target and subdomain using Route 53

1. Sign in to the Route 53 console. If prompted, enter your AWS credentials.
2. If you don't have a public hosted zone in Route 53, create one with a root that is a parent of your custom domain. For more information, see Creating a public hosted zone in the *Amazon Route 53 Developer Guide*.

   a. Choose **Create Hosted Zone**.

   b. Enter the parent domain, for example *auth.example.com*, of your custom domain, for example *myapp.auth.example.com*, from the **Domain Name** list.

   c. Enter a **Description** for your hosted zone.

   d. Choose a hosted zone **Type** of **Public hosted zone** to allow public clients to resolve your custom domain. Choosing **Private hosted zone** is not supported.

   e. Apply **Tags** as desired.

   f. Choose **Create hosted zone**.

   > **Note**
   >
   > You can also create a new hosted zone for your custom domain with a delegation set in the parent hosted zone that directs queries to the subdomain hosted zone. Otherwise, create an A record. This method offers more flexibility and security with your hosted zones.For more information, see Creating a subdomain for a domain hosted through Amazon Route 53.

3. On the **Hosted Zones** page, choose the name of your hosted zone.
4. Add a DNS record for the parent domain of your custom domain, if you don’t already have one. Create a DNS record for the parent domain with the following properties:

   - **Record name**: Leave blank.
   - **Record type**: A.
   - **Alias**: Don't enable.
   - **Value**: Enter a target of your choosing. This record must resolve to *something*, but the value of the record doesn't matter to Amazon Cognito.

- **TTL**: Set to your preferred TTL or leave as default.
- **Routing policy**: Choose **Simple routing**.

5. Choose **Create records**. The following is an example record for the domain *example.com*:

```
example.com. 60 IN A 198.51.100.1
```

> **Note**
>
> Amazon Cognito verifies that there is a DNS record for the parent domain of your custom domain to protect against accidental hijacking of production domains. If you do not have a DNS record for the parent domain, Amazon Cognito will return an error when you attempt to set the custom domain. A Start of Authority (SOA) record isn't a sufficient DNS record for the purposes of parent-domain verification.

6. Add another DNS record for your custom domain with the following properties:

- **Record name**: Your custom domain prefix, for example `auth` to create a record for `auth.example.com`.
- **Record type**: A.
- **Alias**: Enable.
- **Route traffic to**: Choose **Alias to Cloudfront distribution**. Enter the **Alias target** you recorded earlier, for example `123example.cloudfront.net`.
- **Routing policy**: Choose **Simple routing**.

7. Choose **Create records**.

> **Note**
>
> Your new records can take around 60 seconds to propagate to all Route 53 DNS servers. You can use the Route 53 GetChange API method to verify that your changes have propagated.

Step 3: Verify your sign-in page

- Verify that the sign-in page is available from your custom domain.

Sign in with your custom domain and subdomain by entering this address into your browser. This is an example URL of a custom domain *example.com* with the subdomain *auth*:

```
https://myapp.auth.example.com/login?
response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callback_url>
```

Changing the SSL certificate for your custom domain

When necessary, you can use Amazon Cognito to change the certificate that you applied to your custom domain.

Usually, this is unnecessary following routine certificate renewal with ACM. When you renew your existing certificate in ACM, the ARN for your certificate remains the same, and your custom domain uses the new certificate automatically.

However, if you replace your existing certificate with a new one, ACM gives the new certificate a new ARN. To apply the new certificate to your custom domain, you must provide this ARN to Amazon Cognito.

After you provide your new certificate, Amazon Cognito requires up to 1 hour to distribute it to your custom domain.

Before you begin

Before you can change your certificate in Amazon Cognito, you must add your certificate to ACM. For more information, see Getting Started in the *AWS Certificate Manager User Guide*. When you add your certificate to ACM, you must choose US East (N. Virginia) as the AWS Region.

You can change your certificate by using the Amazon Cognito console or API.

AWS Management Console

To renew a certificate from the Amazon Cognito console:

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cognito/home.

2. Choose **User Pools**.
3. Choose the user pool for which you want to update the certificate.
4. Choose the **Domain** menu.
5. Choose **Actions**, **Edit ACM certificate**.
6. Select the new certificate you want to associate with your custom domain.
7. Choose **Save changes**.

API

To renew a certificate (Amazon Cognito API)

- Use the UpdateUserPoolDomain action.

Setting the minimum TLS version for your custom domain

When you create or update a custom domain with the Amazon Cognito console or API, you can select the minimum TLS version for the Amazon CloudFront distribution. The distribution uses this version when it communicates with clients. In the API, set the minimum TLS version in the `SecurityPolicy` field of the `CustomDomainConfig` object in a CreateUserPoolDomain or UpdateUserPoolDomain request. The security policy defines the minimum TLS version and cipher suites that CloudFront supports when communicating with clients. For specific guidance, see Supported protocols and ciphers between viewers and CloudFront in the *Amazon CloudFront Developer Guide*. You can select one of the following values.

`TLS_V1_3_2025` (strictest)

A post-quantum-ready policy requiring TLS 1.3. It provides the strongest security posture and is ideal for workloads where all clients and browsers are updated to the latest versions. For more information, see Supported protocols and ciphers for TLSv1.3_2025.

`TLS_V1_2_2021` (recommended)

A post-quantum-ready policy that prefers TLS 1.3 but allows fallback to TLS 1.2 to accommodate older clients. It is the recommended minimum for typical commercial-grade consumer applications. For more information, see Supported protocols and ciphers for TLSv1.2_2021.

TLS_V1 (strongly discouraged)

Permits fallback to TLS 1.0. It offers the broadest compatibility, including support for legacy clients that are more than a decade old. This compatibility comes at the expense of allowing TLS versions and cryptographic algorithms that are no longer considered safe for commercial use. For more information, see Supported protocols and ciphers for TLSv1.

> **Note**
>
> Custom domains that use managed login (branding version 2) require a minimum TLS version of `TLS_V1_2_2021` or higher. A request that sets `TLS_V1` for a managed login domain returns an `InvalidInputException` error.

You can select the minimum TLS version from the Amazon Cognito console, or with the Amazon Cognito API, the AWS Command Line Interface (AWS CLI), and the AWS SDKs.

The following UpdateUserPoolDomain request body updates the minimum TLS version of an existing custom domain.

```
{
   "Domain": "auth.example.com",
   "UserPoolId": "us-east-1_EXAMPLE",
   "ManagedLoginVersion": 2,
   "CustomDomainConfig": {
    "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/
a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
    "SecurityPolicy": "TLS_V1_3_2025"
   }
}
```

Apply branding to managed login pages

You might want to provide a consistent user experience between your authentication service and your application. You can accomplish this goal either with custom forms and back-end API operations in an AWS SDK, or with managed login. Managed login and the classic hosted UI are web front ends for the component of your application that serves authentication with user pools. To synchronize your managed authentication services with your application UX, you have two

customization options: the branding editor and hosted UI branding. You can choose your preferred experience in the Amazon Cognito console and with user pool API operations.

The branding editor

The branding editor is the newest customization option for the newest user pools UI experience, managed login. The branding editor is a no-code visual editor for managed login assets and style, and a set of API operations for programmatic configuration of a large number of configuration options. User pools that you configure with a domain and managed login automatically render the branding-designer version of your login pages.

Hosted UI (classic) branding

The hosted UI (classic) branding experience has two options: to modify a cascading stylesheets (CSS) file with a fixed set of style options, and to add a custom logo image. You can set these options in the Amazon Cognito console or with the SetUICustomization API operation. At the time that the service launched, Amazon Cognito had only this option. User pools that you configure with a domain and the hosted UI branding version automatically render the classic version of your login pages. Your feature plan might also support only the hosted UI.

> **Note**
>
> The branding editor and the classic branding experience modify the visual properties of your hosted authentication service. Currently, you can't modify the text that's displayed on your managed login pages, except to apply localization into one of several languages. For more information about localization, see Managed login localization.

Choose a branding experience and assign styles

In the Amazon Cognito console, new user pools default to the **Managed login** branding experience. User pools that you set up before managed login was available will have **Hosted UI (classic)** branding. You can switch between managed login and hosted UI branding. When you change your **Branding version**, Amazon Cognito immediately applies the change to the user-interactive pages of your user pool domain. With managed login and the hosted UI, your user pool can have a style for each app client.

Each app client can have a distinct branding *style*, but a user pool domain serves either managed login or the hosted UI. A style is the set of customization settings applied to an app client. You can

set up one custom domain and one prefix domain per user pool. You can assign different branding versions to your custom and prefix domains. However, a prefix domain isn't fully functional when you also have a custom domain—the `.well-known` OIDC discovery endpoints *only* present custom-domain paths. You can only use the prefix domain for operations that don't require endpoint discovery (`openid-configuration`) in a user pool with this configuration. Because of these properties of user pools, you can effectively choose one branding version per user pool.

You can assign styles to the app clients in a user pool where a domain is set to the managed login branding version. Styles are a set of visual settings made up of image files, display options, CSS values. When you assign a style to an app client, Amazon Cognito immediately pushes your updates to your user-interactive login pages. Amazon Cognito renders your user-interactive pages with your chosen branding version and the customization that you have applied to it.

Update and delete styles

When you create a style, you link it to an app client. To change a style assignment for an app client, you must first delete the original style. Currently, you can't copy settings between styles. You must do this programmatically. To replicate settings between styles and app clients, get the settings for a style with the DescribeManagedLoginBranding API operation and apply them with CreateManagedLoginBranding or UpdateManagedLoginBranding. You can't change the assigned styles of an app client—you can only delete the original and set a new one. For more information about managing styles with API and SDK operations, see API and SDK operations for managed login branding.

> **Note**
>
> Programmatic requests that create or update branding style must have a request size of no more than 2 MB. If your request is larger than this limit, break your request up into multiple `UpdateManagedLoginBranding` requests for groups of parameters that don't exceed the maximum request size. These requests don't result in unspecified parameters being set to default, so you can send partial requests without any effect on existing settings.

You delete a style in the Amazon Cognito console from the **Managed login** menu. Under **Styles**, choose the style that you want to delete and choose **Delete style**.

At a high level, the process of assigning branding to a domain consists of the following steps.

1. Create a domain and set the branding version.

2. Create a branding style and assign it to an app client.

To assign a style to an app client

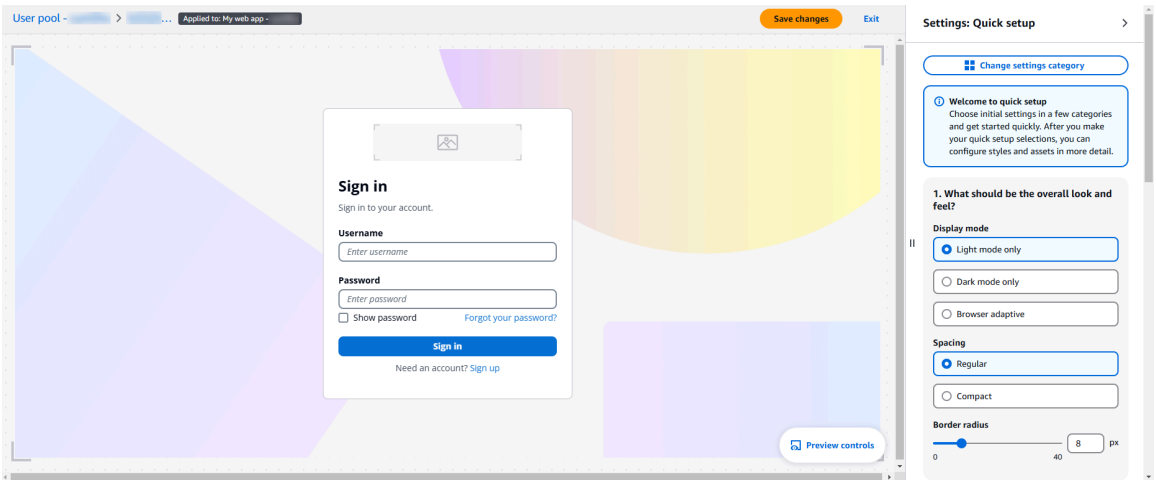
1. In the **Domain** menu of your user pool, create a domain and set the **Branding version** to **Managed login**.
2. Navigate to the **Managed login** menu. Under **Styles**, choose **Create a style**.
3. Choose the app client that you want to assign your style to, or create a new app client.
4. To start configuring your branding settings, choose **Launch branding editor**.

Topics

- The branding editor and customizing managed login

The branding editor and customizing managed login

The branding editor is a visual design and editing tool for your managed login webpages. It's built in to the Amazon Cognito console. In the branding editor, you start with a preview of your login pages and can proceed into a quick-setup option or a detailed view with advanced options. You can modify and preview style parameters or add a custom background image and logo. You can configure light mode and dark mode.



To begin, create a style that you can apply to your user pool or an app client.

To get started with the branding editor

1. Create a domain from the **Domain** tab, or update your existing domain. Under **Branding version**, set your domain to use **Managed login**.
2. Delete the existing app client style, if any.
   a. In the **App clients** menu, select your app client.
   b. Under **Managed login style**, select the syle assigned to your app client.
   c. Choose **Delete style**. Confirm your selection.
3. Navigate to the **Managed login** menu in your user pool. If you haven't already, follow the prompt to select a feature plan that includes managed login. You can also select **Preview this feature** if you want to check out the branding editor without making changes.
4. Under **Styles**, choose **Create a style**.
5. Choose the app client that you want to assign your style to and select **Create**. You can also create a new app client.
6. The Amazon Cognito console launches the branding editor.
7. Choose a tab where you want to start editing, or select **Launch editor** and enter quick setup. The following tabs are available:

   Preview

   See how your current selections look in your managed login pages.

   Foundation

   Set an overall theme, configure links to external identity providers, and style form fields.

   Components

   Configure styles for headers, footers, and individual UI elements.
8. To make choices about initial settings, enter quick setup. Select **Change settings category** and choose **Quick setup**. When you select **Proceed**, the branding editor launches with a set of basic options for you to configure.

Text and localization

You can't modify or localize text in the branding editor. Instead, add a `lang` query parameter to the URL that you distribute to users. This parameters causes your managed login pages to

be localized into one of several available languages. For more information, see Managed login localization.

Quick setup

The **Launch branding editor** button loads a visual editor for your managed login configuration where you can select from a variety of primary customization options. As you make selections, Amazon Cognito renders your managed login changes in a preview window. To return to the detailed settings menu, select the **Change settings category** button.

What should be the overall look and feel?

Configure basic theme settings for managed login.

Display mode

Choose a light-mode, dark-mode, or adaptive experience for your managed login. The adaptive settings defers to the user's browser preference when Amazon Cognito renders managed login. When you choose a browser-adaptive mode, you can choose different colors and logo images for light and dark mode.

Spacing

Set the default spacing between elements in the page.

Border radius

Set the rounding depth of the outer border of elements.

How should the page background look?

Background type

The **Show image** checkbox indicates whether you want a background image or to set a solid background color.

1. To use an image, select **Show image** and choose a background image for light and dark modes. You can also set a dark-mode and light-mode **Page background color** for areas of the background that aren't covered by the image.
2. To use only a color for the background, deselect **Show image** and choose a light-mode and dark-mode **Page background color**.

How should forms look?

Configure settings for the form elements of managed login. Examples of form items are login and code prompts.

Horizontal alignment

Set the horizontal alignment of form fields.

Form logo

Set the positioning of your logo image.

Logo image

Choose a logo image file to include in the form element for light and dark modes. To upload an image, select the **Logo image** dropdown, choose **Add new asset**, and add a logo file.

Primary branding color

Set a theme color for light and dark modes. This color will be applied as the background color to all elements classified as primary.

How should headers look?

Choose whether you want to include a header in your managed login pages. The header can contain a logo image.

Header logo

Set the position of the logo image in your header.

Logo image

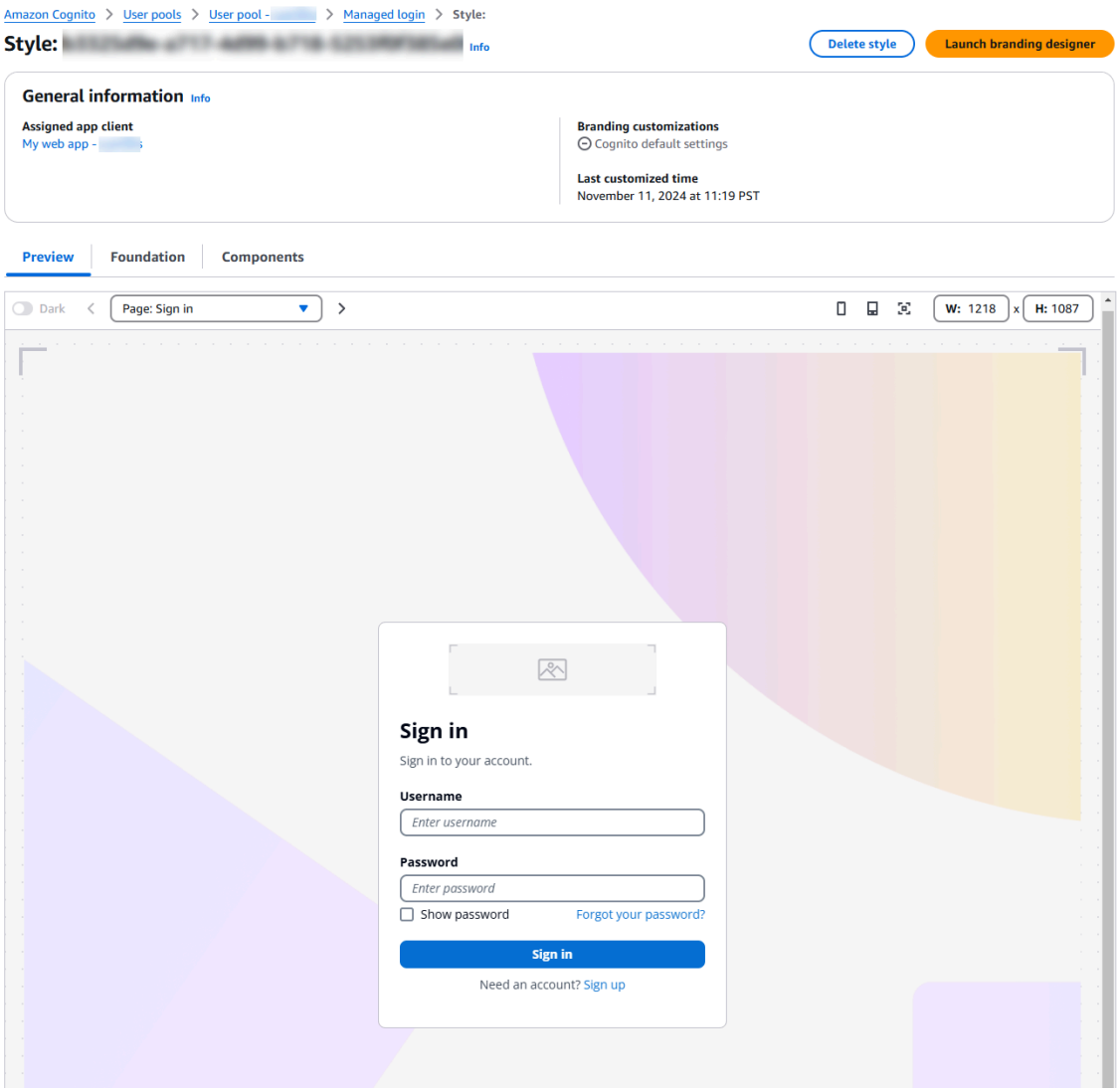
Choose a logo position and a logo image file to include in the header. To upload an image, select the **Logo image** dropdown, choose **Add new asset**, and add a logo file.

Header background color

Set the light and dark mode colors for the background of the header.

Detailed settings

In the detailed settings view, you can modify individual components in the **Foundation** and **Components**. The **Preview** tab displays a preview of managed login in the current context with your customizations.



To enter the visual editor for a component, choose the edit icon in the tile for the component. From the theme studio editor, you can switch between components with the **Change setting category** button.

Foundation

App style

Configure the basics of your managed login configuration. This category has settings for the overall theme, text spacing, and the page header and footer.

Display mode

Choose a light-mode, dark-mode, or adaptive experience for your managed login pages. When you choose a browser-adaptive mode, you can choose different colors and logo images for light and dark mode.

Spacing

Set the default spacing between elements in the page.

Authentication behavior

Configure styles for the buttons that connect your users to external identity providers (IdPs). This section includes the option **Domain search input** to have managed login prompt users for an email address and match them with their SAML identity provider identifier.

Form behavior

Configure styles for input forms: the positioning of inputs, colors, and alignment of elements.

Components

Buttons

Styles for buttons that Amazon Cognito renders on managed login pages.

Divider

Styles for divider lines and boundaries between managed login elements like the input form and the external-provider sign-in selector.

Dropdown

Styles for dropdown menus.

Favicon

Styles for the image that Amazon Cognito provides for the tab and bookmark icon.

Focus rings

Styles for the highlights that indicate a currently-selected input.

Form container

Styles for the elements that bound a form.

Global footer

Styles for the footer that Amazon Cognito displays at the bottom of managed login pages.

Global header

Styles for the header that Amazon Cognito displays at the top of managed login pages.

Indications

Styles for error and success messages.

Option controls

Styles for checkboxes, multi-selects, and other input prompts.

Page background

Styles for the overall background of managed login.

Inputs

Styles for form-field input prompts.

Link

Styles for hyperlinks in managed login pages.

Text for page

Styles for in-page text.

Text for field

Styles for the text around form inputs.

API and SDK operations for managed login branding

You can also apply branding to a managed login style with the API operations CreateManagedLoginBranding and UpdateManagedLoginBranding. These operations are ideal for creating identical or slightly-modified versions of a branding style for another app client or user pool. Query the managed login branding of an existing style with the API operation DescribeManagedLoginBranding, then modify the output as needed and apply it to another resource.

The `UpdateManagedLoginBranding` operation doesn't change the app client that your style is applied to. It only updates the existing style that's assigned to an app client. To completely replace the style for an app client, delete the existing style with DeleteManagedLoginBranding and assign a new style with `CreateManagedLoginBranding`. In the Amazon Cognito console, the same is true: you must delete the existing style and create a new one.

Setting up managed login branding in an API or SDK request requires that your settings be embedded in a JSON file that's converted to a `Document` datatype. The following is guidance for images that you can add and for generating programmatic requests to configure a branding style.

Image assets

CreateManagedLoginBranding and UpdateManagedLoginBranding include an `Assets` parameter. This parameter is an array of image files in base64-encoded binary format.

> **Note**
>
> Programmatic requests that create or update branding style must have a request size of no more than 2 MB. The assets in your request might make it exceed this limit. If this is the case, break your request up into multiple `UpdateManagedLoginBranding` requests for groups of parameters that don't exceed the maximum request size. These requests don't result in unspecified parameters being set to default, so you can send partial requests without any effect on existing settings.

Some assets have limitations on the filetypes that you can submit.

Asset	Accepted file extensions
FAVICON_ICO	ico
FAVICON_SVG	svg
EMAIL_GRAPHIC	png, svg, jpeg
SMS_GRAPHIC	png, svg, jpeg
AUTH_APP_GRAPHIC	png, svg, jpeg
PASSWORD_GRAPHIC	png, svg, jpeg

Asset	Accepted file extensions
PASSKEY_GRAPHIC	png, svg, jpeg
PAGE_HEADER_LOGO	png, svg, jpeg
PAGE_HEADER_BACKGROUND	png, svg, jpeg
PAGE_FOOTER_LOGO	png, svg, jpeg
PAGE_FOOTER_BACKGROUND	png, svg, jpeg
PAGE_BACKGROUND	png, svg, jpeg
FORM_BACKGROUND	png, svg, jpeg
FORM_LOGO	png, svg, jpeg
IDP_BUTTON_ICON	ico, svg

Files of the SVG type support the following attributes and elements.

Attributes

```
accent-height, accumulate, additivive, alignment-baseline, ascent, attributename,
 attributetype, azimuth, basefrequency, baseline-shift, begin, bias, by, class,
 clip, clip-path, clip-rule, color, color-interpolation, color-interpolation-
filters, color-profile, color-rendering, cx, cy, d, dx, dy, diffuseconstant,
 direction, display, divisor, dur, edgemode, elevation, end, fill, fill-opacity,
 fill-rule, filter, filterunits, flood-color, flood-opacity, font-family, font-
size, font-size-adjust, font-stretch, font-style, font-variant, font-weight, fx,
 fy, g1, g2, glyph-name, glyphref, gradientunits, gradienttransform, height, href,
 id, image-rendering, in, in2, k, k1, k2, k3, k4, kerning, keypoints, keysplines,
 keytimes, lang, lengthadjust, letter-spacing, kernelmatrix, kernelunitlength,
 lighting-color, local, marker-end, marker-mid, marker-start, markerheight,
 markerunits, markerwidth, maskcontentunits, maskunits, max, mask, media,
 method, mode, min, name, numoctaves, offset, operator, opacity, order, orient,
 orientation, origin, overflow, paint-order, path, pathlength, patterncontentunits,
 patterntransform, patternunits, points, preservealpha, preserveaspectratio, r,
 rx, ry, radius, refx, refy, repeatcount, repeatdur, restart, result, rotate,
 scale, seed, shape-rendering, specularconstant, specularexponent, spreadmethod,
 stddeviation, stitchtiles, stop-color, stop-opacity, stroke-dasharray, stroke-
```

```
dashoffset, stroke-linecap, stroke-linejoin, stroke-miterlimit, stroke-opacity,
 stroke, stroke-width, style, surfacescale, tabindex, targetx, targety, transform,
 text-anchor, text-decoration, text-rendering, textlength, type, u1, u2, unicode,
 values, viewbox, visibility, vert-adv-y, vert-origin-x, vert-origin-y, width, word-
spacing, wrap, writing-mode, xchannelselector, ychannelselector, x, x1, x2, xmlns,
 y, y1, y2, z, zoomandpan
```

Elements

```
svg, a, altglyph, altglyphdef, altglyphitem, animatecolor, animatemotion,
 animatetransform, audio, canvas, circle, clippath, defs, desc, ellipse, filter,
 font, g, glyph, glyphref, hkern, image, line, lineargradient, marker, mask,
 metadata, mpath, path, pattern, polygon, polyline, radialgradient, rect, stop,
 style, switch, symbol, text, textpath, title, tref, tspan, video, view, vkern,
 feBlend, feColorMatrix, feComponentTransfer, feComposite, feConvolveMatrix,
 feDiffuseLighting, feDisplacementMap, feDistantLight, feFlood, feFuncA, feFuncB,
 feFuncG, feFuncR, feGaussianBlur, feMerge, feMergeNode, feMorphology, feOffset,
 fePointLight, feSpecularLighting, feSpotLight, feTile, feTurbulence
```

Tools for managed login branding operations

Amazon Cognito manages a file in the JSON-Schema format for the managed-login branding settings object. The following is how to programmatically update your branding style.

To update branding in the user pools API

1. In the Amazon Cognito console, create a default managed login branding style from the **Managed login** menu of your user pool. Assign it to an app client.
2. Record the ID of the app client that you created the style for, for example `1example23456789`.
3. Retrieve the settings for the branding style with a DescribeManagedLoginBrandingByClient API request with `ReturnMergedResources` set to `true`. The following is an example request body.

```
{
   "ClientId": "1example23456789",
   "ReturnMergedResources": true,
   "UserPoolId": "us-east-1_EXAMPLE"
}
```

4. Modify the output of `DescribeManagedLoginBrandingByClient` with your customizations.

   a. The response body is wrapped in a `ManagedLoginBranding` element that isn't part of the syntax for create and update operations. Remove this top level of the JSON object.

   b. To replace images, replace the `Bytes` value with the Base64-encoded binary data of each image file.

   c. To update settings, modify the output of the `Settings` object and include it in your next request. Amazon Cognito ignores any values in your `Settings` object that aren't in the schema that you receive in your API response.

5. Use the updated response body in a CreateManagedLoginBranding or UpdateManagedLoginBranding request. If this request exceeds 2 MB in size, separate it out into multiple requests. These operations work in a PATCH model where original settings remain unchanged unless you specify otherwise.

Migrating from the classic hosted UI to managed login

If your user pool currently uses the classic hosted UI, you can migrate it to managed login. Managed login and the classic hosted UI are the two branding versions. You set the branding version on the user pool domain, and your choice applies to all app clients that use that domain, so migrating switches every app client on the domain at the same time. Managed login adds a visual branding editor, passkey sign-in, and additional authentication features. The classic hosted UI uses file-based branding and is available in all feature plans. For a feature-by-feature comparison, see the section called “Managed login”.

> **Important**
>
> Managed login and its branding editor require the Essentials or Plus feature plan. The classic hosted UI is available in all plans, including Lite. Confirm your feature plan before you migrate. See the section called “Managed login”.

At a high level, moving an existing user pool from the classic hosted UI to managed login involves the following steps. Review each step against your current configuration before you switch:

1. Confirm your user pool is on a feature plan that supports managed login. See the section called “Managed login”.

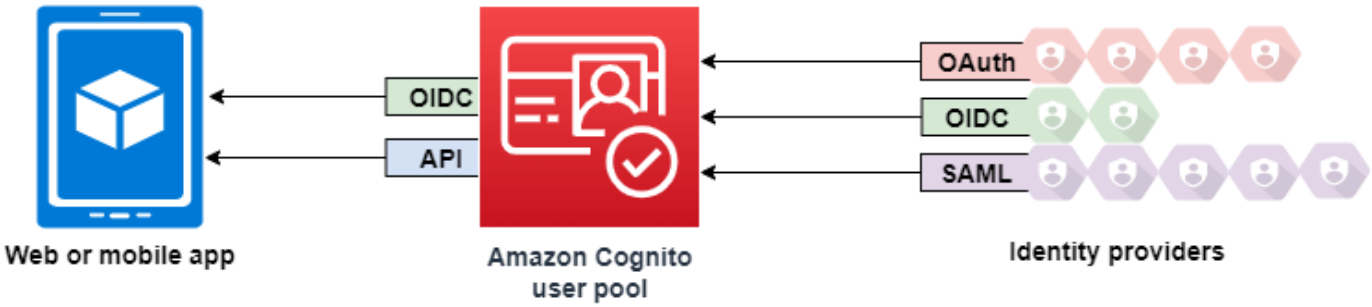
2. Set the branding version on your user pool domain to managed login. The branding version is a domain setting, so it applies to all app clients that use that domain.
3. Set up your branding in the visual branding editor. Managed login uses a different branding model from the classic hosted UI's file-based CSS and logo customization, so review your existing branding against the branding editor as part of the switch. See the section called "Branding and customization".
4. Test the sign-in, sign-up, multi-factor authentication (MFA), and password-reset flows on the managed login pages before you direct production traffic to them.

> **Note**
>
> This page summarizes the migration at a conceptual level. The detailed setup and branding procedures it refers to are the authoritative source for each step.

Authenticating users with Amazon Cognito user pools

An Amazon Cognito user pool is a user directory for web and mobile app authentication and authorization. From the perspective of your app, an Amazon Cognito user pool is an OpenID Connect (OIDC) identity provider (IdP). A user pool adds layers of additional features for security, identity federation, app integration, and customization of the user experience.

This chapter covers how your users authenticate: signing up, signing in, and the authentication flows a user pool supports, including sign-in through external identity providers. Related tasks are covered in their own chapters—managing the users in your directory, customizing behavior with AWS Lambda triggers, and protecting the pool with threat protection.



Topics

- Features
- User pool feature plans
- Authentication with Amazon Cognito user pools
- Accessing resources after successful sign-in

Features

Amazon Cognito user pools have the following features.

Sign-up

Amazon Cognito user pools have user-driven, administrator-driven, and programmatic methods to add user profiles to your user pool. Amazon Cognito user pools supports the following sign-up models. You can use any combination of these models in your app.

> **Important**
>
> If you activate user sign-up in your user pool, anyone on the internet can sign up for an account and sign into your apps. Don't enable self-registration in your user pool unless you want to open your app to public sign-up. To change this setting, update **Self-service sign-up** in the **Sign-up** menu under **Authentication** in the user pool console, or update the value of AllowAdminCreateUserOnly in a CreateUserPool or UpdateUserPool API request. For information about security features that you can set up in your user pools, see Protecting user pools with threat protection.

1. Your users can enter their information in your app and create a user profile that's native to your user pool. You can call API sign-up operations to register users in your user pool. You can open these sign-up operations to anyone, or you can authorize them with a client secret or AWS credentials.
2. You can redirect users to a third-party IdP that they can authorize to pass their information to Amazon Cognito. Amazon Cognito processes OIDC id tokens, OAuth 2.0 `userInfo` data, and SAML 2.0 assertions into user profiles in your user pool. You control the attributes that you want Amazon Cognito to receive based on attribute-mapping rules.
3. You can skip public or federated sign-up and create users yourself, based on your own data source and schema, in any of these ways:
   - Add users directly in the Amazon Cognito console or with the API.
   - Import users in bulk from a CSV file.
   - Run a just-in-time AWS Lambda function that looks up each new user in an existing directory and populates their profile from that data.

After your users sign up, you can add them to groups that Amazon Cognito lists in the access and ID tokens. You can also link user pool groups to IAM roles when you pass the ID token to an identity pool.

Related topics

- Managing users in your user pool
- Understanding API, OIDC, and managed login pages authentication
- Code examples for Amazon Cognito Identity Provider using AWS SDKs

Sign-in

Amazon Cognito can be a standalone user directory and identity provider (IdP) to your app. Your users can sign in with managed login pages that are hosted by Amazon Cognito, or with a custom-built user authentication service through the Amazon Cognito user pools API. The application tier behind your custom-built front end can authorize requests on the back end with any of several methods to confirm legitimate requests.

Users can set up and sign with usernames and passwords, passkeys, and email and SMS message one-time passwords. You can offer consolidate sign in with external user directories, multi-factor authentication (MFA) after sign-in, trust remembered devices, and custom authentication flows that you design.

To sign in users with an external directory, optionally combined with the user directory built in to Amazon Cognito, you can add the following integrations.

1. Sign in and import customer user data with OAuth 2.0 social sign-in. Amazon Cognito supports sign-in with Google, Facebook, Amazon, and Apple through OAuth 2.0.
2. Sign in and import work and school user data with SAML and OIDC sign-in. You can also configure Amazon Cognito to accept claims from any SAML or OpenID Connect (OIDC) identity provider (IdP).
3. Link external user profiles to native user profiles. A linked user can sign in with a third-party user identity and receive access that you assign to a user in the built-in directory.

Related topics

- User pool sign-in with third party identity providers
- Linking federated users to an existing user profile

Machine-to-machine authorization

Some sessions aren't a human-to-machine interaction. You might need a service account that can authorize a request to an API by an automated process. To generate access tokens for machine-to-machine authorization with OAuth 2.0 scopes, you can add an app client that generates client-credentials grants from the token endpoint, or call the `GetClientToken` API operation, which requires no user pool domain.

Related topics

- Scopes, M2M, and resource servers

Managed login

When you don't want to build a user interface, you can present your users with a customized managed login pages. Managed login is a set of web pages for sign-up, sign-in, multi-factor authentication (MFA), and password reset. You can add managed login to your existing domain, or use a prefix domain in an AWS subdomain.

Managed login has an alternative: the classic hosted UI, a first-generation version with a simpler design and fewer features. You choose one or the other for a user pool domain—they aren't used together. The following table compares the two options.

Feature	Classic hosted UI	Managed login
Branding version	Hosted UI (classic)	Managed login
Customization	File-based branding (logo image and a CSS values file)	Visual branding editor in the Amazon Cognito console
Feature plans	All plans, including Lite	Essentials and Plus
Passkey sign-in	Not available	Available

> **Note**
>
> The classic hosted UI is available in all feature plans, including Lite. Managed login and its branding editor require the Essentials or Plus feature plan. For more information, see User pool feature plans.

To set up managed login, complete these prerequisites in order:

1. Create a user pool. See Create a new application in the Amazon Cognito console.
2. Add a user pool domain (a prefix domain or a custom domain). See Configuring a user pool domain.

3. Create an app client and set its branding version and return URL. See User pool managed login.

Related topics

- User pool managed login
- Configuring a user pool domain

Threat protection

Protect your users against password guessing, account takeover, and malicious traffic with multi-factor authentication (MFA), adaptive authentication, and AWS WAF web ACLs. These protections are covered in Protecting user pools with threat protection.

Custom user experience

At most stages of sign-up, sign-in, or profile update, you can customize how Amazon Cognito handles the request with Lambda triggers and custom authentication flows, and give managed login a familiar look and feel. See Customizing user pool workflows with Lambda triggers.

Monitoring and analytics

Amazon Cognito user pools log API requests to AWS CloudTrail, publish performance metrics to Amazon CloudWatch Logs, and monitor message delivery and request volume. See Amazon Cognito logging in AWS CloudTrail.

Amazon Cognito identity pools integration

The other half of Amazon Cognito is identity pools. Identity pools provide credentials that authorize and monitor API requests to AWS services, for example Amazon DynamoDB or Amazon S3, from your users. You can build identity-based access policies that protect your data based on how you classify the users in your user pool. Identity pools can also accept tokens and SAML 2.0 assertions from a variety of identity providers, independently of user pool authentication.

Related topics

- Accessing AWS services using an identity pool after sign-in
- Amazon Cognito identity pools

User pool feature plans

Understanding the cost is a crucial step in preparing to implement Amazon Cognito user pools authentication. Amazon Cognito has feature plans for user pools. Each plan has a set of features and a monthly cost per active user. Each feature plan unlocks access to more features than the one before it.

User pools have a variety of features that you can turn on and off. For example, you can turn on multi-factor authentication (MFA) and turn off sign-in with third-party identity providers (IdPs). Some changes require you to switch your feature plan. The following characteristics of your user pool determine the cost that AWS bills you monthly for usage.

- The features that you choose
- The requests per second that your application makes to the user pools API
- The number of users with authentication, update, or query activity in a month, also called monthly active users or MAUs
- The number of monthly active users from third-party SAML 2.0 or OpenID Connect (OIDC) IdPs
- The number of app clients and user pools that do client-credentials grants for machine-to-machine authorization

For the most current information about user pool pricing, see Amazon Cognito pricing.

Feature-plan selections apply to one user pool. Different user pools in the same AWS account can have different plan selections. You can't apply separate feature plans to app clients within a user pool. The default plan selection for new user pools is Essentials.

You can switch between feature plans at any time to fit the requirements of your applications. Some changes between plans require that you turn off active features. For more information, see Turning off features to change feature plans.

User pool feature plans

Lite

Lite is a low-cost feature plan for user pools with lower numbers of monthly active users. This plan is sufficient for user directories with basic authentication features. It includes sign-in features and the classic hosted UI, a slimmer, less-customizable predecessor to managed

login. Many newer features, like access-token customization and passkey authentication, aren't included in the Lite plan.

Essentials

Essentials has all of the latest user pool authentication features. This plan adds new options to your applications, whether your login pages are managed login or custom-built. Essentials has advanced authentication features like choice-based sign-in and email MFA.

Plus

Plus includes everything in the Essentials plan and adds advanced security features that protect your users. Monitor user sign-in, sign-up, and password-management requests for indicators of compromise. For example, user pools can detect whether users are signing in from an unexpected location or using a password that's been part of a public breach.

User pools with the Plus plan generate logs of user activity details and risk evaluations. You can apply your own usage and security analysis to these logs when you export them to external services.

> **Note**
>
> Previously, some user pool features were included in an *advanced security features* pricing structure. The features that were included in this structure are now under either the Essentials or Plus plan.

Topics

- Select a feature plan
- Features by plan
- Essentials plan features
- Plus plan features
- Turning off features to change feature plans

Select a feature plan

AWS Management Console

To choose a feature plan

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Select the **Settings** menu and review the **Feature plans** tab.
5. Review the features available to you in the Lite, Esssentials, and Plus plans.
6. To change your plan, select **Switch to Essentials**, or **Switch to Plus**. To switch to the **Lite** plan, choose **Other plans**, then **Compare with Lite**.
7. On the next screen, review your choice and select **Confirm**.

CLI/API/SDK

The CreateUserPool and UpdateUserPool operations set your feature plan in the `UserPoolTier` parameter. When you don't specify a value for `UserPoolTier`, your user pool defaults to `Essentials`. If you set `AdvancedSecurityMode` to `AUDIT` or `ENFORCED`, your user pool tier must be `PLUS` and default to `PLUS` when not specified.

See Examples in CreateUserPool for syntax. See See Also in CreateUserPool for links to this function in of AWS SDKs for a variety of programming languages.

```
"UserPoolTier": "PLUS"
```

In the AWS CLI, this option is `--user-pool-tier` argument.

```
--user-pool-tier PLUS
```

See create-user-pool and update-user-pool in the AWS CLI command reference for more information.

Features by plan

Features and plans in user pools

Feature	Description	Feature plan
Protect against unsafe passwords	Check plaintext passwords for indicators of compromise at runtime	Plus
Protect against malicious sign-in attempts	Check session properties for indicators of compromise at runtime	Plus
Log and analyze user activity	Generate logs of user authentication session properties and risk scores	Plus
Export user activity logs	Push user session and risk logs to an external AWS service	Plus
Customize managed login pages with a visual editor	Use a visual editor in the Amazon Cognito console to apply branding and style to your managed login pages	Essentials + Plus
MFA with email one-time codes	Request or require local users to provide an additional email message sign-in factor after username authentication	Essentials + Plus
Customize access token scopes and claims at runtime	Use a Lambda trigger to extend the authorization capabilities of user pool access tokens	Essentials + Plus
Passwordless sign-in with one-time codes	Permit users to receive a one-time password by email or	Essentials + Plus

Feature	Description	Feature plan
	SMS as their first authentic ation factor	
Passkey sign-in with hardware or software FIDO2 authentic ators	Permit users to use a cryptographic key stored on a FIDO2 authenticator as their first authentication factor	Essentials + Plus
Sign-up and sign-in	Perform authentication operations and let new users register for an account in your application.	Lite + Essentials + Plus
User groups	Create logical groupings of users and assign default IAM roles for identity pool operations.	Lite + Essentials + Plus
Sign-in with social, SAML, and OIDC providers	Provide users with the options to sign in directly or with their preferred provider.	Lite + Essentials + Plus
OAuth 2.0/OIDC authoriza tion server	Act as a OIDC issuer.	Lite + Essentials + Plus
Login pages	A hosted collection of webpages for authentication. Managed login is available in the Essentials and Plus tiers. The classic hosted UI is available in all feature tiers.	Lite + Essentials + Plus
Password, custom, refresh-t oken, and SRP authentication	Prompt users for a username and password in your application.	Lite + Essentials + Plus

Feature	Description	Feature plan
Machine-to-machine (M2M) with client credentials	Issue access tokens for authorization of non-human entities.	Lite + Essentials + Plus
API authorization with resource servers	Issue access tokens with custom scopes that authorize access to external systems.	Lite + Essentials + Plus
User import	Set up import jobs from CSV files and perform just-in-time migration of users as they sign in.	Lite + Essentials + Plus
MFA with authenticator apps and SMS one-time codes	Request or require local users to provide an additional SMS message or authenticator app sign-in factor after username authentication	Lite + Essentials + Plus
Customize ID token scopes and claims at runtime	Use a Lambda trigger to extend the authentication capabilities of user pool identity (ID) tokens	Lite + Essentials + Plus
Custom runtime actions with Lambda triggers	Customize the sign-in process at runtime with Lambda functions that perform external actions and influence authentication	Lite + Essentials + Plus
Customize managed login pages with CSS	Download a CSS template and change some styles in your managed login pages	Lite + Essentials + Plus

Essentials plan features

The Essentials feature plan has most of the best and latest features of Amazon Cognito user pools. When you switch from the Lite to the Essentials plan, you get new features for your managed login pages, multi-factor authentication with email-message one-time passwords, an enhanced password policy, and custom access tokens. To stay up-to-date with new user pool features, choose the Essentials plan for your user pools.

The sections that follows present a brief overview of the features that you can add to your application with the Essentials plan. For detailed information, see the following pages.

Additional resources

- Access token customization: Pre token generation Lambda trigger
- Email MFA: SMS and email message MFA
- Password history: Passwords, account recovery, and password policies
- Enhanced UI: Apply branding to managed login pages

Topics

- Access token customization
- Email MFA
- Password reuse prevention
- Managed login hosted sign-in and authorization server
- Choice-based authentication

Access token customization

User pool access tokens grant permissions to applications: to access an API, to retrieve user attributes from the userInfo endpoint, or to establish group membership for an external system. In advanced scenarios, you might want to add to the default access-token data from the user pool directory with additional temporary parameters that your application determines at runtime. For example, you might want to verify a user's API permissions with Amazon Verified Permissions and adjust the scopes in the access token accordingly.

The Essentials plan adds to the existing functions of a pre token generation trigger. With lower-tier plans, you can customize ID tokens with additional claims, roles, and group membership. Essentials

adds new versions of the trigger input event that customize access token claims, roles, group membership, and scopes. Access token customization is available to machine-to-machine (M2M) client credentials grants with event version three.

To customize access tokens

1. Select the Essentials or Plus feature plan.
2. Create a Lambda function for your trigger. To use our example function, configure it for Node.js.
3. Populate your Lambda function with our example code or compose your own. You function must process a request object from Amazon Cognito and return the changes that you want to include.
4. Assign your new function as a version two or three pre token generation trigger. Version two events customize access tokens for user identities. Version three customizes access tokens for user and machine identities.

Learn more

- Customizing the access token
- How to customize access tokens in Amazon Cognito user pools

Email MFA

Amazon Cognito user pools can be configured to use email as the second factor in multi-factor authentication (MFA). With email MFA, Amazon Cognito can send users an email with a verification code that they must enter to complete the authentication process. This adds an important extra layer of security to the user login flow. To enable email-based MFA, the user pool must be configured to use the Amazon SES email-sending configuration instead of the default email configuration.

When your user selects MFA by email message, Amazon Cognito will send a one-time verification code to the user's registered email address whenever they attempt to sign in. The user must then provide this code back to your user pool to complete the authentication flow and gain access. This ensures that even if a user's username and password are compromised, they must provide an additional factor—the emailed code—before they can access your application resources.

For more information, see SMS and email message MFA. The following is an overview of how to set up your user pool and users for email MFA.

To set up email MFA in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.
2. In the **Sign-in** menu of your user pool, edit **Multi-factor authentication**.
3. Choose the level of **MFA enforcement** that you want to set up. With **Require MFA**, users in the API automatically receive a challenge to set up, confirm, and sign in with MFA. In user pools that require MFA, managed login prompts them to choose and set up an MFA factor. With **Optional MFA**, your application must offer users the option to set up MFA and set the user's preference for email MFA.
4. Under **MFA methods**, select **Email message** as one of the options.

Learn more

- SMS and email message MFA

Password reuse prevention

By default, a Amazon Cognito user pools password policy sets password length and character-type requirements, and temporary-password expiration. The Essentials plan adds the capability to enforce password history. When a user attempts to reset their password, your user pool can prevent them from setting it to a previous password. For more information about configuring the password policy, see Adding user pool password requirements. The following is an overview of how to set up your user pool with a password-history policy.

To set up password history in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.
2. In the **Authentication methods** menu of your user pool, locate **Password policy** and select **Edit**.
3. Configure other available options and set a value for **Prevent use of previous passwords**.

Learn more

- Passwords, account recovery, and password policies

Managed login hosted sign-in and authorization server

Amazon Cognito user pools have optional webpages that support the following functions: an OpenID Connect (OIDC) IdP, a service provider or relying party to third-party IdPs, and public user-interactive pages for sign-up and sign-in. These pages are collectively called *managed login*. When you choose a domain for your user pool, Amazon Cognito automatically activates these pages. Where the Lite plan has the hosted UI, the Essentials plan opens up this advanced version of sign-up and sign-in pages.

Managed login pages have a clean, up-to-date interface with more features and options for customizing your branding and styles. The Essentials plan is the lowest plan level that unlocks access to managed login.

To set up managed login in the Amazon Cognito console

1. From the **Settings** menu, select the Essentials or Plus feature plan.
2. From the **Domain** menu, Assign a domain to your user pool and select a **Branding version** of **Managed login**.
3. From the **Managed login** menu, under **Styles** tab, choose **Create a style** and assign the style to an app client, or create a new app client.

Learn more

- User pool managed login

Choice-based authentication

The Essentials tier introduces a new *authentication flow* for authentication operations in the enhanced UI and SDK-based API operations.This flow is *choice-based authentication*. Choice-based authentication is a method where your users' authentication starts not with an application-side declaration of a sign-in method, but a query of possible sign-in methods followed by a choice. You can configure your user pool to support choice-based authentication and unlock username-password, passwordless, and passkey authentication. In the API, this is the `USER_AUTH` flow.

To set up choice-based authentication in the Amazon Cognito console

1. Select the Essentials or Plus feature plan.

2. In the **Sign-in** menu of your user pool, edit **Options for choice-based sign-in**. Select and configure the authentication methods you want to enable in choice-based authentication.
3. In the **Authentication methods** menu of your user pool, edit the configuration of sign-in operations.

Learn more

- Authentication with Amazon Cognito user pools

Plus plan features

The Plus feature plan has advanced security features for Amazon Cognito user pools. These features log and analyze user context at runtime for potential security issues in devices, locations, request data, and passwords. They then mitigate potential risks with automatic responses that block or add security safeguards to user accounts. You can also export your security logs to Amazon S3, Amazon Data Firehose, or Amazon CloudWatch Logs for further analysis.

When you switch from the Essentials to the Plus plan, you get all the features in Essentials and the additional features that follow. These include the threat protection set of security options also known as *advanced security features*. To configure your user pools to automatically adapt to threats in your authentication front end, choose the Plus plan for your user pools.

The sections that follows present a brief overview of the features that you can add to your application with the Plus plan. For detailed information, see the following pages.

Additional resources

- Adaptive authentication: Working with adaptive authentication
- Compromised credentials: Working with compromised-credentials detection
- Log export: Exporting logs from Amazon Cognito user pools

Topics

- Threat protection: adaptive authentication
- Threat protection: compromised-credentials detection
- Threat protection: user activity logging

Threat protection: adaptive authentication

The Plus plan includes an *adaptive authentication* feature. When you activate this feature, your user pool makes a risk assessment of every user authentication session. From the resulting risk ratings, you can block authentication or push MFA for users who sign in with a risk level above a threshhold that you determine. With adaptive authentication, your user pool and application automatically block or set up MFA for users whose accounts you suspect are being attacked. You can also provide feedback on the risk ratings from your user pool to adjust future ratings.

To set up adaptive authentication in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Full-function**.
4. Under **Adaptive authentication**, configure automatic risk responses for different levels of risk.

Learn more

- Working with adaptive authentication
- Collecting data for threat protection in applications

Threat protection: compromised-credentials detection

The Plus plan includes a *compromised-credentials detection* feature. This feature guards against the use of insecure passwords and the threat of unintended application access that this practice creates. When you permit your users to sign in with username and password, they might reuse a password that they've used elsewhere. That password might have been leaked, or just be commonly guessed. With compromised-credentials detection, your user pool reads the passwords your users submit and compares them to password databases. If the operation results in a decision that the password is likely compromised, you can configure your user pool to block sign-in and then initiate a password reset for the user in your application.

Compromised-credentials detection can react to insecure passwords when new users sign up, when existing users sign in, and when users attempt to reset their passwords. With this feature, your user pool can prevent or warn about sign-in with insecure passwords wherever users enter them.

To set up compromised-credentials detection in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Full-function**.
4. Under **Compromised credentials**, configure the types of authentication operations that you want to check, and the automated response that you want from your user pool.

Learn more

- Working with compromised-credentials detection

Threat protection: user activity logging

The Plus plan adds a logging feature that gives security analysis and details of user authentication attempts. You can see risk assessments, user IP addresses, user agents, and other information about the device that connected to your application. You can act on this information with the built-in threat protection features, or you can analyze your logs in your own systems and take appropriate action. You can export the logs from threat protection to Amazon S3, CloudWatch Logs, or Amazon DynamoDB.

To set up user activity logging in the Amazon Cognito console

1. Select the Plus feature plan.
2. From the **Threat protection** menu of your user pool, edit **Standard and custom authentication** under **Threat protection**.
3. Set the **enforcement mode** for standard or custom authentication to **Audit-only**. This is the minimum setting for logs. You can also activate it in **Full-function** mode and configure other threat protection features.
4. To export your logs to another AWS service for third-party analysis, go to the **Log streaming** menu of your user pool and set up an export destination.

Learn more

- Exporting user authentication events

- Exporting logs from Amazon Cognito user pools

Turning off features to change feature plans

Feature plans add configuration options to your user pool. You can configure and use these features only when the related feature plan is active. For example, you can configure access token customization in the Plus and Essentials plans, but not in the Lite plan. To deactivate these features, you must deactivate each active component. The **Switch to** option in the **Settings** menu in the Amazon Cognito console notifies you of the features you must deactivate before you can change your feature plan. With this chapter, you can learn the changes that deactivation makes to your user pool configuration, and how to turn off these features individually.

Access token customization

To switch to a plan that doesn't include access token customization, you must remove the pre token generation Lambda trigger from your user pool. To add a new pre token generation trigger without access token customization, assign a new function to the trigger and configure it for `V1_0` events. These version one trigger events can process changes to ID tokens only.

To manually deactivate access token customization, remove your pre token generation trigger and add a new version one trigger.

Threat protection

To switch to a plan without threat protection, deactivate all features from the **Threat protection** menu of your user pool.

Log export

To switch to a plan without log export, deactivate it from the **Log streaming** menu of your user pool. Your user pool no longer generates local or exported user-activity logs. You can also send a SetLogDeliveryConfiguration API request that removes any configuration with an `EventSource` value of `UserActivity`.

Email MFA

To switch to a plan without email MFA, go to the **Sign-in** menu of your user pool. Edit **Multi-factor authentication** and deselect **Email message** as one of the available **MFA methods**.

Authentication with Amazon Cognito user pools

Amazon Cognito includes several methods to authenticate your users. Users can sign in with passwords and WebAuthn passkeys. Amazon Cognito can send them a one-time password in an email or SMS message. You can implement Lambda functions that orchestrate your own sequence of challenges and responses. These are *authentication flows*. In authentication flows, users provide a secret and Amazon Cognito verifies the secret, then issues JSON web tokens (JWTs) for applications to process with OIDC libraries. In this chapter, we'll talk about how to configure your user pools and app clients for various authentication flows in various application environments. You'll learn about options for the use of the hosted sign-in pages of managed login, and for building your own logic and front end in an AWS SDK.

All user pools, whether you have a domain or not, can authenticate users in the user pools API. If you add a domain to your user pool, you can use the user pool endpoints. The user pools API supports a variety of authorization models and request flows for API requests.

To verify the identity of users, Amazon Cognito supports authentication flows that incorporate challenge types in addition to passwords like email and SMS message one-time passwords and passkeys.

Before you configure the details, choose the authentication approach that fits your application. The following decision aid summarizes the two entry points and, within a custom-built application, the two initial sign-in flows. Each option links to its detailed section.

Do you want Amazon Cognito to host the sign-in pages, or build your own front end?

- If you want the lowest-effort path, use managed login. Amazon Cognito hosts the sign-in, sign-out, and password-reset pages, and your application processes the result with an OpenID Connect (OIDC) relying-party library. Managed login automatically presents the authentication methods that your user pool and app client configuration allow.

- If you want to build your own UI and control the sign-in logic, use an AWS SDK. Your application calls the user pools API directly and you implement the challenge-response logic.

In a custom-built application, do you collect the sign-in method from the user, or declare it up front?

- Choose choice-based authentication (the USER_AUTH flow) when you want to offer users a list of available sign-in methods and let them select one. This flow is the only way to offer passwordless and passkey sign-in.

Choose client-based authentication when your application already knows how the user will sign in and declares the flow up front, for example USER_SRP_AUTH. Client-based authentication is the only way to use the custom authentication and refresh token flows.

Prerequisites for choice-based authentication

Choice-based authentication applies to both managed login and the AWS SDKs. To use it, add ALLOW_USER_AUTH to the allowed authentication flows of your app client. The passwordless and passkey methods additionally require a managed login domain and a feature plan above the **Lite** tier (Essentials or Plus).

Topics

- Application-specific settings with app clients
- Implement authentication flows
- Things to know about authentication with user pools
- An example authentication session
- Configure authentication methods for managed login
- Manage authentication methods in AWS SDKs
- Authentication flows
- Authorization models for API and SDK authentication
- User pool sign-in with third party identity providers

Application-specific settings with app clients

A user pool app client is a configuration within a user pool that interacts with one mobile or web application that authenticates with Amazon Cognito. App clients can call authenticated and unauthenticated API operations, and read or modify some or all of your users' attributes. Your app must identify itself to the app client in operations to register, sign in, and handle forgotten passwords. These API requests must include self-identification with an app client ID, and authorization with an optional client secret. You must secure any app client IDs or secrets so that only authorized client apps can call these unauthenticated operations. Additionally, if you configure your app to sign authenticated API requests with AWS credentials, you must secure your credentials against user inspection.

You can create multiple apps for a user pool. An app client might be linked to the code platform of an app, or a separate tenant in your user pool. For example, you might create an app for a server-side application and a different Android app. Each app has its own app client ID.

You can apply settings for the following user pool features at the app client level:

1. Analytics
2. Managed login IdPs, grant types, callback URLs, and customization
3. Resource servers and custom scopes
4. Threat protection
5. Attribute read and write permissions
6. Token expiration and revocation
7. Authentication flows

App client types

When you create an app client in Amazon Cognito, you can pre-populate options based on the standard OAuth client types **public client** and **confidential client**. Configure a **confidential client** with a **client secret**. For more information about client types, see IETF RFC 6749 #2.1.

Public client

A public client runs in a browser or on a mobile device. Because it does not have trusted server-side resources, it does not have a client secret.

Confidential client

A confidential client has server-side resources that can be trusted with a **client secret** for unauthenticated API operations. The app might run as a daemon or shell script on your backend server.

Client secret

A client secret, or client password, is a fixed string that your app must use in all API requests to the app client. Your app client must have a client secret to perform `client_credentials` grants. For more information, see IETF RFC 6749 #2.3.1.

Each app client can have up to two secrets at a time, enabling secret rotation without downtime. When you create an app client, you can either let Amazon Cognito generate a secret

value or provide your own custom secret value. You can't change secrets after you create an app. You can add a second secret with the AddUserPoolClientSecret API operation to rotate secrets. When you add a secret, you can either let Amazon Cognito generate a secret value or provide your own custom secret value. To delete a secret, use the DeleteUserPoolClientSecret API operation. You cannot delete the only secret associated with an app client. You can also delete an app to block access from apps that use that app client ID.

> **Note**
>
> The Amazon Cognito console creates app clients with client secrets when you select the **Traditional web application** and **Machine-to-machine application** options for application type. Choose one of these options to generate a client secret, or create the client programmatically with CreateUserPoolClient and set `GenerateSecret` to `true`.

> **Important**
>
> You can't change whether an app client has a client secret after you create the app client. The UpdateUserPoolClient operation can't add or remove a client secret. To change this configuration, create a new app client. You don't need to recreate the user pool, and your other app clients aren't affected.

You can use a confidential client, and a client secret, with a public app. Use an Amazon CloudFront proxy to add a `SECRET_HASH` in transit. For more information, see Protect public clients for Amazon Cognito by using an Amazon CloudFront proxy on the AWS blog.

JSON web tokens

Amazon Cognito app clients can issue JSON web tokens (JWTs) of the following types.

Identity (ID) token

A verifiable statement that your user is authenticated from your user pool. OpenID Connect (OIDC) added the ID token specification to the access and refresh token standards defined by OAuth 2.0. The ID token contains identity information, like user attributes, that your app can use to create a user profile and provision resources. See Understanding the identity (ID) token for more information.

Access token

A verifiable statement of your user's access rights. The access token contains scopes, a feature of OIDC and OAuth 2.0. Your app can present scopes to back-end resources and prove that your user pool authorized a user or machine to access data from an API, or their own user data. An access token with *custom scopes*, often from an M2M client-credentials grant, authorizes access to a resource server. See Understanding the access token for more information.

Refresh token

An encrypted statement of initial authentication that your app can present to your user pool when your user's tokens expire. A refresh-token request returns new, unexpired access and ID tokens. See Refresh tokens for more information.

You can set the expiration of these tokens for each app client from the **App clients** menu of your user pool in the Amazon Cognito console.

App client terms

The following terms are available properties of app clients in the Amazon Cognito console.

Allowed callback URLs

A callback URL indicates where the user will be redirected after a successful sign-in. Choose at least one callback URL. The callback URL must:

- Be an absolute URI.
- Be pre-registered with a client.
- Not include a fragment component.

See OAuth 2.0 - redirection endpoint.

Amazon Cognito requires `HTTPS` over `HTTP` except for `http://localhost` for testing purposes only.

App callback URLs such as `myapp://example` are also supported.

Allowed sign out URLs

A sign-out URL indicates where your user is to be redirected after signing out.

Attribute read and write permissions

Your user pool might have many customers, each with their own app client and IdPs. You can configure your app client to have read and write access to only those user attributes that are relevant to the app. In cases like machine-to-machine (M2M) authorization, you can grant access to none of your user attributes.

Considerations for attribute read and write permissions configuration

- When you create an app client and don't customize attribute read and write permissions, Amazon Cognito grants read and write permissions to all user pool attributes.
- You can grant write access to immutable custom attributes. Your app client can write values to immutable attributes when you create or sign up a user. After this, you can't write values to any immutable custom attributes for the user.
- App clients must have write access to required attributes in your user pool. The Amazon Cognito console automatically sets required attributes as writeable.
- You can't permit an app client to have write access to `email_verified` or `phone_number_verified`. A user pool administrator can modify these values. A user can only change the value of these attributes through attribute verification.

Authentication flows

The methods that your app client allows for sign-in. Your app can support authentication with username and password, email and SMS message OTPs, passkey authenticators, custom authentication with Lambda triggers, and token refresh. As a best security practice, use SRP authentication for username and password authentication in custom-built applications.

Custom scopes

A custom scope is one that you define for your own resource server in the **Resource Servers**. The format is *`resource-server-identifier`*/*`scope`*. See Scopes, M2M, and resource servers.

Default redirect URI

Replaces the `redirect_uri` parameter in authentication requests for users with third-party IdPs. Configure this app client setting with the `DefaultRedirectURI` parameter of a CreateUserPoolClient or UpdateUserPoolClient API request. This URL must also be a member of the `CallbackURLs` for your app client. Amazon Cognito redirects authenticated sessions to this URL when:

1. Your app client has one identity provider assigned and multiple callback URLs defined. Your user pool redirects authentication requests to the authorization server to the default redirect URI when they don't include a `redirect_uri` parameter.
2. Your app client has one identity provider assigned and one callback URLs defined. In this scenario it's not necessary to define a default callback URL. Requests that don't include a `redirect_uri` parameter redirect to the one available callback URL.

Identity providers

You can choose some or all of your user pool external identity providers (IdPs) to authenticate your users. Your app client can also authenticate only local users in your user pool. When you add an IdP to your app client, you can generate authorization links to the IdP and display it on your managed login sign-in page. You can assign multiple IdPs, but you must assign at least one. For more information on using external IdPs, see User pool sign-in with third party identity providers.

OpenID Connect scopes

Choose one or more of the following `OAuth` scopes to specify the access privileges that can be requested for access tokens.

- The `openid` scope declares that you want to retrieve an ID token and a user's unique ID. It also requests all or some user attributes, depending on additional scopes in the request. Amazon Cognito doesn't return an ID token unless you request the `openid` scope. The `openid` scope authorizes structural ID token claims like expiration and key ID, and determines the user attributes that you receive in a response from the userInfo endpoint.
  - When `openid` is the only scope that you request, Amazon Cognito populates the ID token with all user attributes that the current app client can read. The `userInfo` response to an access token with this scope alone returns all user attributes.
  - When you request `openid` with other scopes like phone, `email`, or `profile`, the ID token and `userInfo` return the user's unique ID and the attributes defined by the additional scopes.
- The phone scope grants access to the `phone_number` and `phone_number_verified` claims. This scope can only be requested with the `openid` scope.
- The `email` scope grants access to the `email` and `email_verified` claims. This scope can only be requested with the `openid` scope.

- The `aws.cognito.signin.user.admin` scope grants access to Amazon Cognito user pools API operations that require access tokens, such as UpdateUserAttributes and VerifyUserAttribute.
- The `profile` scope grants access to all user attributes that are readable by the client. This scope can only be requested with the `openid` scope.

For more information about scopes, see the list of standard OIDC scopes.

OAuth grant types

An OAuth grant is a method of authentication that retrieves user-pool tokens. Amazon Cognito supports the following types of grants. To integrate these OAuth grants in your app, you must add a domain to your user pool.

Authorization code grant

The authorization code grant generates a code that your app can exchange for user pool tokens with the Token endpoint. When you exchange an authorization code, your app receives ID, access, and refresh tokens. This OAuth flow, like the implicit grant, happens in your users' browsers. An authorization code grant is the most secure grant that Amazon Cognito offers, because tokens aren't visible in your users' sessions. Instead, your app generates the request that returns tokens and can cache them in protected storage. For more information, see *Authorization code* in IETF RFC 6749 #1.3.1

> **Note**
>
> As a best security practice in public-client apps, activate only the authorization-code grant OAuth flow, and implement Proof Key for Code Exchange (PKCE) to restrict token exchange. With PKCE, a client can only exchange an authorization code when they have provided the token endpoint with the same secret that was presented in the original authentication request. For more information on PKCE, see IETF RFC 7636.

Implicit grant

The implicit grant delivers an access and ID token, but not refresh token, to your user's browser session directly from the Authorize endpoint. An implicit grant removes the requirement for a separate request to the token endpoint, but isn't compatible with PKCE and doesn't return refresh tokens. This grant accommodates testing scenarios and app architecture that can't

complete authorization-code grants. For more information, see *Implicit grant* in IETF RFC 6749 #1.3.2. You can activate both the authorization-code grant and the implicit grant in an app client, and then use each grant as needed.

Client credentials grant

The client credentials grant is for machine-to-machine (M2M) communications. Authorization-code and implicit grants issue tokens to authenticated human users. Client credentials grant scope-based authorization from a non-interactive system to an API. Your app can request client credentials directly from the token endpoint and receive an access token. For more information, see *Client Credentials* in IETF RFC 6749 #1.3.4. You can only activate client-credentials grants in app clients that have a client secret and that don't support authorization-code or implicit grants. As an alternative that requires no user pool domain, an app client with a client secret and the `ALLOW_CLIENT_TOKEN_AUTH` authentication flow can obtain M2M access tokens with the GetClientToken API operation.

> **Note**
>
> Because you don't invoke the client credentials flow as a user, this grant can only add *custom* scopes to access tokens. A custom scope is one that you define for your own resource server. Default scopes like `openid` and `profile` don't apply to nonhuman users.
> Because ID tokens are a validation of user attributes, they aren't relevant to M2M communication, and a client credentials grants doesn't issue them. See Scopes, M2M, and resource servers.

Client credentials grants add costs to your AWS bill. For more information, see Amazon Cognito Pricing.

Creating an app client

AWS Management Console

To create an app client (console)

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.

3. Choose an existing user pool from the list, or create a user pool. Both options prompt you to configure an app client with application-specific settings.
4. Choose an **Application type** that reflects your application architecture.

   > **Important**
   >
   > Your **Application type** selection also sets defaults for other app client settings, including whether Amazon Cognito generates a client secret. The **Traditional web application** and **Machine-to-machine application** types generate a client secret. For more information, see App client types.

5. **Name your application** with a friendly identifier.
6. Enter a **Return URL**.
7. Choose **Create app client**. You can change advanced options after you create your app client.
8. Amazon Cognito returns you to app client details. To access example code for your application, select a platform from the **Quick setup guide** tab.

AWS CLI

```
aws cognito-idp create-user-pool-client --user-pool-id MyUserPoolID --client-
name myApp
```

> **Note**
>
> Use JSON format for callback and logout URLs to prevent the CLI from treating them as remote parameter files:
>
> ```
> --callback-urls "["https://example.com"]"
> --logout-urls "["https://example.com"]"
> ```

See the AWS CLI command reference for more information: create-user-pool-client

Amazon Cognito user pools API

Generate a CreateUserPoolClient API request. You must specify a value for all parameters that you don't want set to a default value.

Updating a user pool app client (AWS CLI and AWS API)

At the AWS CLI, enter the following command:

```
aws cognito-idp update-user-pool-client --user-pool-id  "MyUserPoolID" --client-id
 "MyAppClientID" --allowed-o-auth-flows-user-pool-client --allowed-o-auth-flows "code"
 "implicit" --allowed-o-auth-scopes "openid" --callback-urls "["https://example.com"]"
 --supported-identity-providers "["MySAMLIdP", "LoginWithAmazon"]"
```

If the command is successful, the AWS CLI returns a confirmation:

```
{
    "UserPoolClient": {
        "ClientId": "MyClientID",
        "SupportedIdentityProviders": [
            "LoginWithAmazon",
            "MySAMLIdP"
        ],
        "CallbackURLs": [
            "https://example.com"
        ],
        "AllowedOAuthScopes": [
            "openid"
        ],
        "ClientName": "Example",
        "AllowedOAuthFlows": [
            "implicit",
            "code"
        ],
        "RefreshTokenValidity": 30,
        "AuthSessionValidity": 3,
        "CreationDate": 1524628110.29,
        "AllowedOAuthFlowsUserPoolClient": true,
        "UserPoolId": "MyUserPoolID",
        "LastModifiedDate": 1530055177.553
    }
}
```

See the AWS CLI command reference for more information: update-user-pool-client.

AWS API: UpdateUserPoolClient

Getting information about a user pool app client (AWS CLI and AWS API)

```
aws cognito-idp describe-user-pool-client --user-pool-id MyUserPoolID --client-
id MyClientID
```

See the AWS CLI command reference for more information: describe-user-pool-client.

AWS API: DescribeUserPoolClient

Listing all app client information in a user pool (AWS CLI and AWS API)

```
aws cognito-idp list-user-pool-clients --user-pool-id "MyUserPoolID" --max-results 3
```

See the AWS CLI command reference for more information: list-user-pool-clients.

AWS API: ListUserPoolClients

Deleting a user pool app client (AWS CLI and AWS API)

```
aws cognito-idp delete-user-pool-client --user-pool-id "MyUserPoolID" --client-id
 "MyAppClientID"
```

See the AWS CLI command reference for more information: delete-user-pool-client

AWS API: DeleteUserPoolClient

Implement authentication flows

Whether you're implementing managed login or a custom-built application front end with an AWS SDK for authentication, you must configure your app client for the types of authentication that you want to implement. The following information describes setup for authentication flows in your app clients and your application.

App client supported flows

You can configure supported flows for your app clients in the Amazon Cognito console or with the API in an AWS SDK. After you configure your app client to support these flows, you can deploy them in your application.

The following procedure configures available authentication flows for an app client with the Amazon Cognito console.

To configure an app client for authentication flows (console)

1. Sign in to AWS and navigate to the Amazon Cognito user pools console. Choose a user pool or create a new one.
2. In your user pool configuration, select the **App clients** menu. Choose an app client or create a new one.
3. Under **App client information**, select **Edit**.
4. Under **App client flows**, choose the authentication flows that you want to support.

To configure an app client for authentication flows (API/SDK)

To configure available authentication flows for an app client with the Amazon Cognito API, set the value of `ExplicitAuthFlows` in a CreateUserPoolClient or UpdateUserPoolClient request. The following is an example that provisions secure remote password (SRP) and choice-based authentication to a client.

```
"ExplicitAuthFlows": [
   "ALLOW_USER_AUTH",
   "ALLOW_USER_SRP_AUTH"
]
```

When you configure app client supported flows, you can specify the following options and API values.

App client flow support

Authentication flow	Compatibility	Console	API
Choice-based authentication	Server-side, client-side	Select an authentication type at sign-in	`ALLOW_USER_AUTH`
Sign-in with persistent passwords	Client-side	Sign in with username and password	`ALLOW_USER_PASSWORD_AUTH`
Sign-in with persistent passwords and secure payload	Server-side, client-side	Sign in with secure remote password (SRP)	`ALLOW_USER_SRP_AUTH`

Authentication flow	Compatibility	Console	API
Refresh tokens	Server-side, client-side	Get new user tokens from existing authenticated sessions	`ALLOW_REFRESH_TOKEN_AUTH`
Server-side authentication	Server-side	Sign in with server-side administrative credentials	`ALLOW_ADMIN_USER_PASSWORD_AUTH`
Custom authentication	Server-side and client-side custom-built applications. Not compatible with managed login.	Sign in with custom authentication flows from Lambda triggers	`ALLOW_CUSTOM_AUTH`

Implement flows in your application

Managed login automatically makes your configured authentication options available in your sign-pages. In custom-built applications, start authentication with a declaration of the initial flow.

- To choose from a list of flow options for a user, declare choice-based authentication with the `USER_AUTH` flow. This flow has available authentication methods that aren't available in client-based authentication flows, for example passkey and passwordless authentication.
- To choose your authentication flow up front, declare client-based authentication with any other flow that's available in your app client.

When you sign users in, the body of your InitiateAuth or AdminInitiateAuth request must include an `AuthFlow` parameter.

Choice-based authentication:

```
"AuthFlow": "USER_AUTH"
```

Client-based authentication with SRP:

```
"AuthFlow": "USER_SRP_AUTH"
```

Things to know about authentication with user pools

Consider the following information in the design of your authentication model with Amazon Cognito user pools.

Authentication flows in managed login and the hosted UI

Managed login has more options for authentication than the classic hosted UI. For example, users can do passwordless and passkey authentication only in managed login.

Custom authentication flows only available in AWS SDK authentication

You can't do *custom authentication flows*, or custom authentication with Lambda triggers, with managed login or the classic hosted UI. Custom authentication is available in authentication with AWS SDKs.

Managed login for external identity provider (IdP) sign-in

You can't sign users in through third-party IdPs in authentication with AWS SDKs. You must implement managed login or the classic hosted UI, redirect to IdPs, and then process the resulting authentication object with OIDC libraries in your application. For more information about managed login, see User pool managed login.

Passwordless authentication effect on other user features

Activation of passwordless sign-in with one-time passwords or passkeys in your user pool and app client has an effect on user creation and migration. When passwordless sign-in is active:

1. Administrators can create users without passwords. The default invitation message template changes to no longer include the {###} password placeholder. For more information, see Creating user accounts as administrator.
2. For SDK-based SignUp operations, users aren't required to supply a password when they sign up. Managed login and the hosted UI require a password in the sign-up page, even if passwordless authentication is permitted. For more information, see Signing up and confirming user accounts.
3. Users imported from a CSV file can sign in immediatelywith passwordless options, without a password reset, if their attributes include an email address or phone number for an available passwordless sign-in option. For more information, see Importing users into user pools from a CSV file.

4. Passwordless authentication doesn't invoke the user migration Lambda trigger.
5. Users who sign in with a one-time password (OTP) first factor can't add a multi-factor authentication (MFA) factor to their session. Passkeys with user verification can satisfy MFA requirements when configured with `MULTI_FACTOR_WITH_USER_VERIFICATION`.

Passkey relying party URLs can't be on the public suffix list

You can use domain names that you own, like `www.example.com`, as the relying party (RP) ID in your passkey configuration. This configuration is intended to support custom-built applications that run on domains that you own. The public suffix list, or PSL, contains protected high-level domains. Amazon Cognito returns an error when you attempt to set your RP URL to a domain on the PSL.

Authentication session flow duration

Depending on the features of your user pool, you can end up responding to several challenges to `InitiateAuth` and `RespondToAuthChallenge` before your app retrieves tokens from Amazon Cognito. Amazon Cognito includes a session string in the response to each request. To combine your API requests into an authentication flow, include the session string from the response to the previous request in each subsequent request. By default, your users have three minutes to complete each challenge before the session string expires. To adjust this period, change your app client **Authentication flow session duration**. The following procedure describes how to change this setting in your app client configuration.

> **Note**
>
> **Authentication flow session duration** settings apply to authentication with the Amazon Cognito user pools API. Managed login sets session duration to 3 minutes for multi-factor authentication and 8 minutes for forced password change during end-user's first sign-in with temporary password.

Amazon Cognito console

To configure app client authentication flow session duration (AWS Management Console)

1. From the **App integration** tab in your user pool, select the name of your app client from the **App clients and analytics** container.
2. Choose **Edit** in the **App client information** container.

3. Change the value of **Authentication flow session duration** to the validity duration that you want, in minutes, for SMS and email MFA codes. This also changes the amount of time that any user has to complete any authentication challenge in your app client.
4. Choose **Save changes**.

User pools API

To configure app client authentication flow session duration (Amazon Cognito API)

1. Prepare an `UpdateUserPoolClient` request with your existing user pool settings from a `DescribeUserPoolClient` request. Your `UpdateUserPoolClient` request must include all existing app client properties.
2. Change the value of `AuthSessionValidity` to the validity duration that you want, in minutes, for SMS MFA codes. This also changes the amount of time that any user has to complete any authentication challenge in your app client.

For more information about app clients, see Application-specific settings with app clients.

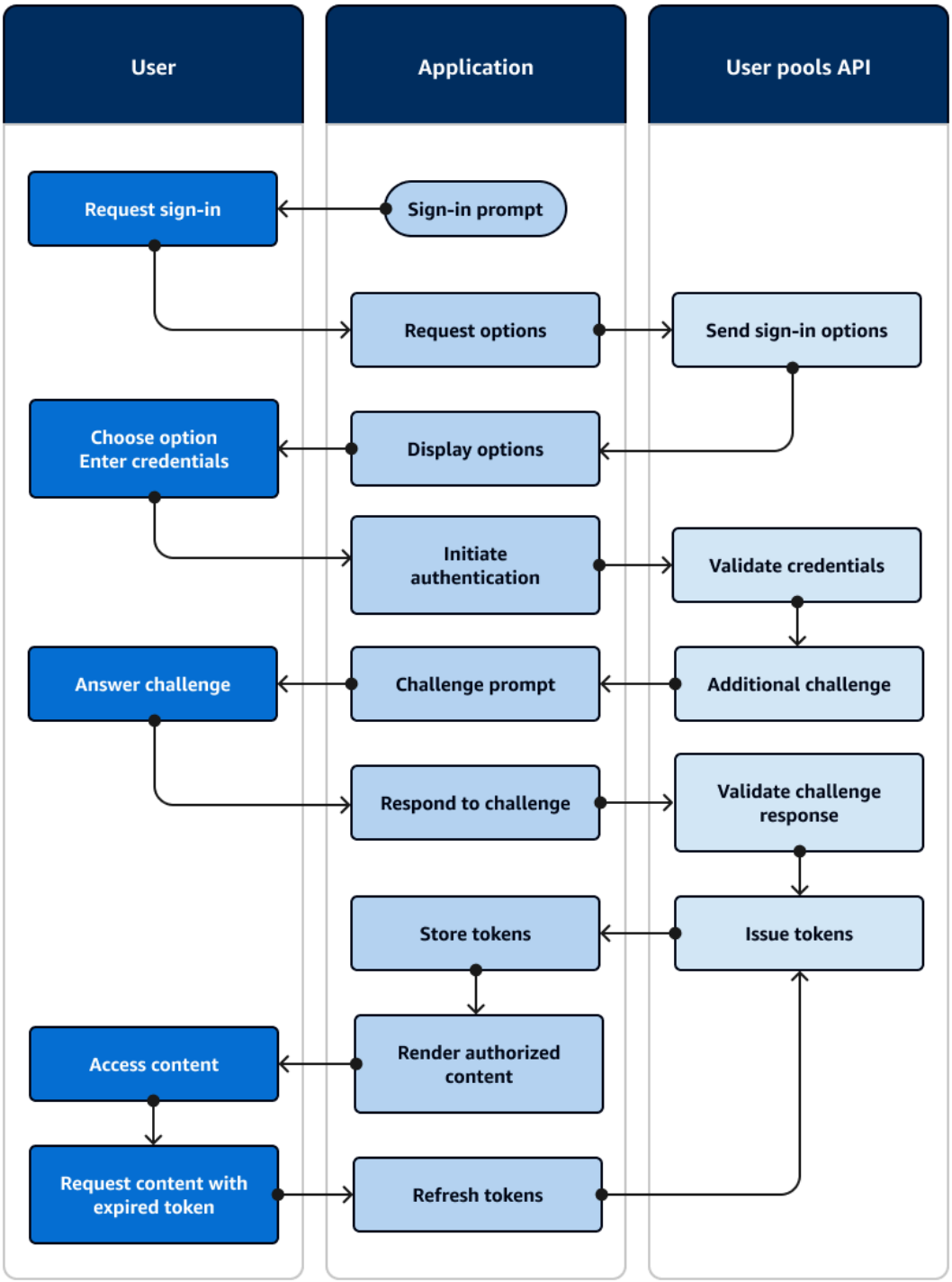
Lockout behavior for failed sign-in attempts

After five failed sign-in attempts with a user's password, regardless of whether those are requested with unauthenticated or IAM-authorized API operations, Amazon Cognito locks out your user for one second. The lockout duration then doubles after each additional one failed attempt, up to a maximum of approximately 15 minutes.

Attempts made during a lockout period generate a `Password attempts exceeded` exception, and don't affect the duration of subsequent lockout periods. For a cumulative number of failed sign-in attempts *n*, not including `Password attempts exceeded` exceptions, Amazon Cognito locks out your user for *2^(n-5)* seconds. To reset the lockout to its *n=0* initial state, your user must either sign in successfully after a lockout period expires, or not initiate any sign-in attempts for 15 consecutive minutes at any time after a lockout. This behavior is subject to change. This behavior doesn't apply to custom challenges unless they also perform password-based authentication.

An example authentication session

The following diagram and step-by-step guide illustrate a typical scenario where a user signs in to an application. The example application presents a user with several sign-in options. They select one by entering their credentials, provide an additional authentication factor, and sign in.



Picture an application with a sign-in page where users can sign in with a username and password, request a one-time code in an email message, or choose a fingerprint option.

1. **Sign-in prompt**: Your application shows a home screen with a *Log in* button.

2. **Request sign-in**: The user selects *Log in*. From a cookie or a cache, your application retrieves their username, or prompts them to enter it.
3. **Request options**: Your application requests the user's sign-in options with an `InitiateAuth` API request with the `USER_AUTH` flow, requesting the available sign-in methods for the user.
4. **Send sign-in options**: Amazon Cognito responds with `PASSWORD`, `EMAIL_OTP`, and `WEB_AUTHN`. The response includes a session identifier for you to replay back in the next response.
5. **Display options**: Your application shows UI elements for the user to enter their username and password, get a one-time code, or scan their fingerprint.
6. **Choose option/Enter credentials**: The user enters their username and password.
7. **Initiate authentication**: Your application provides the user's sign-in information with a `RespondToAuthChallenge` API request that confirms username-password sign-in and provides the username and the password.
8. **Validate credentials**: Amazon Cognito confirms the user's credentials.
9. **Additional challenge**: The user has multi-factor authentication configured with an authenticator app. Amazon Cognito returns a `SOFTWARE_TOKEN_MFA` challenge.
10. **Challenge prompt**: Your application displays a form requesting a time-based one-time password (TOTP) from the user's authenticator app.
11. **Answer challenge**: The user submits the TOTP.
12. **Respond to challenge**: In another `RespondToAuthChallenge` request, your application provides the user's TOTP.
13. **Validate challenge response**: Amazon Cognito confirms the user's code and determines that your user pool is configured to issue no additional challenges to the current user.
14. **Issue tokens**: Amazon Cognito returns ID, access, and refresh JSON web tokens (JWTs). The user's initial authentication is complete.
15. **Store tokens**: Your application caches the user's tokens so that it can reference user data, authorize access to resources, and update tokens when they expire.
16. **Render authorized content**: Your application makes a determination of the user's access to resources based on their identity and roles, and delivers application content.
17. **Access content**: The user is signed in and begins using the application.
18. **Request content with expired token**: Later, the user requests a resource that requires authorization. The user's cached token has expired.
19. **Refresh tokens**: Your application makes an `InitiateAuth` request with the user's saved refresh token.

20 **Issue tokens**: Amazon Cognito returns new ID and access JWTs. The user's session is securely refreshed without additional prompts for credentials.

You can use AWS Lambda triggers to customize the way users authenticate. These triggers issue and verify their own challenges as part of the authentication flow.

You can also use the admin authentication flow for secure backend servers. You can use the user migration authentication flow to make user migration possible without the requirement that your users to reset their passwords.

Configure authentication methods for managed login

You can invoke managed login pages, a web front end for user pool authentication, when you want users to sign in, sign out, or reset their password. In this model, your application imports OIDC libraries to process browser-based authentication attempts with user pool managed login pages. The forms of authentication that are available to your users are dependent on the configuration of your user pool and your app client. Implement the `ALLOW_USER_AUTH` flow in your app client, and Amazon Cognito prompts users to select a sign-in method from the available options. Implement `ALLOW_USER_PASSWORD_AUTH` and assign a SAML provider, and your login pages prompt users with the option to enter their username and password or to connect with their IdP.

The Amazon Cognito user pools console can get you started with setting up managed login authentication for your application. When you create a new user pool, specify the platform you're developing for and the console gives you examples for implementation of OIDC and OAuth libraries with starter code to implement sign-in and sign-out flows. You can build managed login with many OIDC relying-party implementations. We recommend that you work with certified OIDC relying party libraries where possible. For more information, see Create a new application in the Amazon Cognito console.

Typically, OIDC relying party libraries periodically check the `.well-known/openid-configuration` endpoint of your user pool to determine issuer URLs like the token endpoint and authorization endpoint. As a best practice, implement this automatic-discovery behavior where you have to option to. Manual configuration of issuer endpoints introduces potential for error. For example, you might change your user pool domain. The path to `openid-configuration` isn't linked to your user pool domain, so applications that autodiscover service endpoints will automatically pick up your domain change.

User pool settings for managed login

You might want to allow sign in with multiple providers for your application, or you might want to use Amazon Cognito as an independent user directory. You might also want to collect user attributes, set up and prompt for MFA, or require email addresses as usernames. You can't directly edit the fields in managed login and the hosted UI. Instead, the configuration of your user pool automatically sets the handling of managed-login authentication flows.

The following user pool configuration items determine the authentication methods that Amazon Cognito presents to users in managed login and the hosted UI.

User pool options (Sign-in menu)

The following options are in the **Sign-in** menu of a user pool in the Amazon Cognito console.

Cognito user pool sign-in options

Has options for usernames. Your managed login and hosted UI pages only accept usernames in the formats that you select. When you, for example, set up a user pool with **Email** as the only sign-in option, your managed login pages only accept usernames in an email format.

Required attributes

When you set an attribute as required in your user pool, managed login prompts users for a value for that attribute when they sign up.

Options for choice-based sign-in

Has settings for authentication methods in Choice-based authentication. Here, you can turn on or off authentication methods like passkey and passwordless. These methods are only available to user pools with managed login domains and feature plans above the **Lite** tier.

Multi-factor authentication

Managed login and the hosted UI handle registration and authentication operations for MFA. When MFA is required in your user pool, your sign-in pages automatically prompt users to set up their additional factor. They also prompt users who have an MFA configuration to complete authentication with an MFA code. When MFA is off or optional in your user pool, your sign-in pages don't prompt to set up MFA.

User account recovery

The self-service account recovery setting of your user pool determines whether your sign-in pages display a link where users can reset their password.

User pool options (Domain menu)

The following options are in the **Domain** menu of a user pool in the Amazon Cognito console.

Domain

Your choice of a user pool domain sets the path for the link that users open when you invoke their browsers for authentication.

Branding version

Your choice of a branding version determines whether your user pool domain displays managed login or the hosted UI.

User pool options (Social and external providers menu)

The following option is in the **Social and external providers** menu of a user pool in the Amazon Cognito console.

Providers

The identity providers (IdPs) that you add to your user pool can be left active or inactive for each app client in the user pool.

App client options

The following options are in the **App clients** menu of a user pool in the Amazon Cognito console. To review these options, select an app client from the list.

Quick setup guide

The quick setup guide has code examples for a variety of developer environments. They contain the libraries necessary to integrate managed login authentication with your application.

App client information

Edit this configuration to set assigned IdPs for the application that's represented by the current app client. On the managed login pages, Amazon Cognito displays choices for users. These choices are determined from the assigned methods and IdP. For example, if you assign a SAML 2.0 IdP named `MySAML` and local user pool login, your managed login pages display authentication-method prompts and a button for `MySAML`.

Authentication settings

Edit this configuration to set authentication methods for your application. On the managed login pages, Amazon Cognito displays choices for users. These choices are determined from the availability of the user pool as an IdP, and from the methods that you assign. For example, if you assign choice-based `ALLOW_USER_AUTH` authentication, your managed login pages display available choices like entering an email address and signing in with a passkey. Managed login pages also render buttons for the assigned IdPs.

Login pages

Set the visual effect of your managed login or hosted UI user-interactive pages with the options available in this tab. For more information, see Apply branding to managed login pages.

Manage authentication methods in AWS SDKs

Users in Amazon Cognito user pools can sign in with a variety of initial sign-in options, or *factors*. For some factors, users can follow up with multi-factor authentication (MFA). These first factors include username and password, one-time password, passkey, and custom authentication. For more information, see Authentication flows. When your application has built-in UI components and imports an AWS SDK module, you must build application logic for authentication. You must choose one of two primary methods and from that method, the authentication mechanisms that you want to implement.

You can implement *client-based authentication* where your application, or client, declares the type of authentication up front. Your other option is *choice-based authentication*, where your app collects a username and requests the available authentication types for users. You can implement these models together in the same application or split between app clients, according to your requirements. Each method has features that are unique to it, for example custom authentication in client-based and passwordless authentication in choice-based.

In custom-built applications that perform authentication with AWS SDK implementation of the users pools API, you must structure your API requests to align with user pool configuration, app client configuration, and client-side preferences. An `InitiateAuth` session that begins with an `AuthFlow` of `USER_AUTH` begins choice-based authentication. Amazon Cognito responds to your API with a challenge of either a preferred authentication method or a list of choices. A session that begins with `AuthFlow` of `CUSTOM_AUTH` goes right into custom authentication with Lambda triggers.

Some authentication methods are fixed to one of the two flow types, and some methods are available in both.

Topics

- Choice-based authentication
- Client-based authentication
- Step-up authentication with the USER_AUTH flow

Choice-based authentication

Your application can request the following authentication methods in choice-based authentication. Declare these options in the `PREFERRED_CHALLENGE` parameter of InitiateAuth or AdminInitiateAuth, or in the `ChallengeName` parameter of RespondToAuthChallenge or AdminRespondToAuthChallenge.

> **Important**
>
> Choice-based sign-in has two prerequisites. Your app client must activate the `ALLOW_USER_AUTH` authentication flow, and the passwordless and passkey methods (`EMAIL_OTP`, `SMS_OTP`, `WEB_AUTHN`) require a user pool on the **Essentials** or **Plus** feature plan—they aren't available on the **Lite** tier. `PASSWORD` (including SRP) is available on all feature plans. If a choice-based method doesn't appear for your users, confirm both the app-client flow and the feature plan before troubleshooting further.

1. `EMAIL_OTP` and `SMS_OTP`

   Passwordless sign-in with one-time passwords
2. `WEB_AUTHN`

   Passwordless sign-in with WebAuthn passkeys
3. `PASSWORD`

   Sign-in with persistent passwords

   Sign-in with persistent passwords and secure payload

   MFA after sign-in

To review these options in their API context, see `ChallengeName` in RespondToAuthChallenge.

Choice-based sign-in issues a challenge in response to your initial request. This challenge either verifies that a requested option is available, or provides a list of available choices. Your application can display these choices to users, who then enter credentials for their preferred sign-in method and proceed with authentication in challenge responses.

You have the following choice-based options in your authentication flow. All requests of this type require that your app first collect a username or retrieve it from a cache.

1. Request options with `AuthParameters` of `USERNAME` only. Amazon Cognito returns a `SELECT_CHALLENGE` challenge. From there, your application can prompt the user to select a challenge and return this response to your user pool.
2. Request a preferred challenge with `AuthParameters` of `PREFERRED_CHALLENGE` and the parameters of your preferred challenge, if any. For example, if you request a `PREFERRED_CHALLENGE` of `PASSWORD_SRP`, you must also include `SRP_A`. If your user, user pool, and app client are all configured for the preferred challenge, Amazon Cognito responds with the next step in that challenge, for example `PASSWORD_VERIFIER` in the `PASSWORD_SRP` flow or CodeDeliveryDetails in the `EMAIL_OTP` and `SMS_OTP` flows. If the preferred challenge isn't available, Amazon Cognito responds with `SELECT_CHALLENGE` and a list of available challenges.
3. Sign users in first, then request their choice-based authentication options. A GetUserAuthFactors request with the access token of a signed-in user returns their available choice-based authentication factors and their MFA settings. With this option, a user can sign in with username and password first, then activate a different form of authentication. You can also use this operation to check additional options for a user who has signed in with a preferred challenge.

To configure your app client for choice-based authentication, add `ALLOW_USER_AUTH` to the allowed authentication flows. You must also choose the choice-based factors that you want to permit in your user pool configuration. The following process illustrates how to choose choice-based authentication factors.

Amazon Cognito console

To configure choice-based authentication options in a user pool

1. Sign in to AWS and navigate to the Amazon Cognito user pools console. Choose a user pool or create a new one.

2. In your user pool configuration, select the **Sign-in** menu. Locate **Options for choice-based sign-in** and choose **Edit**.
3. The **Password** option is always available. This includes the `PASSWORD` and `PASSWORD_SRP` flows. Select the **Additional choices** that you want to add to your users' options. You can add **Passkey** for `WEB_AUTHN`, **Email message one-time password** for `EMAIL_OTP`, and **SMS message one-time password** for `SMS_OTP`.
4. Choose **Save changes**.

API/SDK

The following partial CreateUserPool or UpdateUserPool request body configures all available options for choice-based authentication.

```
"Policies": {
    "SignInPolicy": {
        "AllowedFirstAuthFactors": [
            "PASSWORD",
            "WEB_AUTHN",
            "EMAIL_OTP",
            "SMS_OTP"
        ]
    }
},
```

Client-based authentication

Client-based authentication supports the following authentication flows. Declare these options in the `AuthFlow` parameter of InitiateAuth or AdminInitiateAuth.

1. `USER_PASSWORD_AUTH` and `ADMIN_USER_PASSWORD_AUTH`

   Sign-in with persistent passwords

   MFA after sign-in

   This authentication flow is equivalent to `PASSWORD` in choice-based authentication.
2. `USER_SRP_AUTH`

   Sign-in with persistent passwords and secure payload

MFA after sign-in

This authentication flow is equivalent to PASSWORD_SRP in choice-based authentication.

3. REFRESH_TOKEN_AUTH

   Refresh tokens

   This authentication flow is only available in client-based authentication.

4. CUSTOM_AUTH

   Custom authentication

   This authentication flow is only available in client-based authentication.

With client-based authentication, Amazon Cognito assumes that you have determined how your user wants to authenticate before they begin authentication flows. The logic of determining the sign-in factor that a user wants to provide must be determined with default settings or custom prompts, then declared in the first request to your user pool. The InitiateAuth request declares a sign-in AuthFlow that directly corresponds to one of the listed options, for example USER_SRP_AUTH. With this declaration, the request also includes the parameters to begin authentication, for example USERNAME, SECRET_HASH, and SRP_A. Amazon Cognito might follow up this request with additional challenges like PASSWORD_VERIFIER for SRP or SOFTWARE_TOKEN_MFA for password sign-in with TOTP MFA.

To configure your app client for client-based authentication, add any authentication flows other than ALLOW_USER_AUTH to the allowed authentication flows. Examples are ALLOW_USER_PASSWORD_AUTH, ALLOW_CUSTOM_AUTH, ALLOW_REFRESH_TOKEN_AUTH. To permit client-based authentication flows, no additional user pool configuration is required.

Step-up authentication with the USER_AUTH flow

You request step-up authentication in the existing USER_AUTH flow with the InitiateAuth and RespondToAuthChallenge operations. There are no new API operations. The corresponding server-side operations, AdminInitiateAuth and AdminRespondToAuthChallenge, support the same step-up parameters and behavior, and AdminInitiateAuth can also optionally take an access token. This flow works both when a user has an existing ID token and when a user authenticates toward a target level for the first time. For more information about the flow, see An example authentication session.

For more information about ACR levels and AMR values, see the section called "Authentication levels with ACR and AMR claims".

> **Note**
>
> Step-up authentication requires the Essentials or Plus feature plan.

You pass the step-up parameters in the `AuthParameters` map of your `InitiateAuth` (or `AdminInitiateAuth`) request: the new `TARGET_ACR_VALUES` and `MAX_AGE`, together with the existing `USERNAME`, `ACCESS_TOKEN`, and `PREFERRED_CHALLENGE`. For the definition and constraints of each parameter, see InitiateAuth and AdminInitiateAuth in the *Amazon Cognito user pools API Reference*.

The following table describes how Amazon Cognito responds to combinations of these parameters.

USER_AUTH step-up authentication behavior

Parameters	Behavior
`TARGET_ACR_VALUES` and `ACCESS_TOKEN`, with `MAX_AGE` satisfied	Amazon Cognito steps the user up from the token's current factors to the highest-priority satisfiable level.
`TARGET_ACR_VALUES` and `ACCESS_TOKEN`, with `MAX_AGE` not satisfied	Amazon Cognito authenticates the user from scratch toward the target level.
`TARGET_ACR_VALUES` without `ACCESS_TOKEN`	Amazon Cognito authenticates the user from scratch toward the target level.
Neither parameter	The `USER_AUTH` flow behaves as it does without step-up authentication.

Keep the following behavior in mind when you design a step-up authentication flow.

- Step-up authentication is all-or-nothing. If reaching the target level requires multiple challenges, for example a password and a TOTP for level 4, and the user fails any challenge, the step-up attempt fails entirely. Amazon Cognito doesn't issue intermediate tokens.

- If the presented token already meets or exceeds the requested level, Amazon Cognito returns an error instead of issuing a new token. This prevents applications from using step-up authentication as a token-refresh mechanism.
- The user must already have the factors to satisfy a requested level. If the user has no factors configured to meet any requested target level, Amazon Cognito returns an error. Step-up authentication doesn't prompt the user to set up new factors.
- MFA is a floor. If the MFA settings of the user or the user pool require MFA, initial sign-in still enforces and prompts that MFA challenge even when the target level is lower and wouldn't otherwise require MFA. MFA that threat protection (advanced security features) prompts for is also enforced on initial sign-in, even when the requested target ACR level wouldn't otherwise require MFA. `TARGET_ACR_VALUES` can't suppress a required MFA prompt. At most, the target can influence which of several enabled MFA factors Amazon Cognito prompts for. If a step-up request targets a lower level, for example level 1, for a user who is required to use MFA, Amazon Cognito silently enforces the required MFA. It steps the user up and doesn't return an error.

Requesting step-up authentication with `TARGET_ACR_VALUES` requires the Essentials or Plus feature plan. For more information, see Feature plan requirements.

Authentication flows

The process of authentication with Amazon Cognito user pools can best be described as a *flow* where users make an initial choice, submit credentials, and respond to additional challenges. When you implement managed login authentication in your application, Amazon Cognito manages the flow of these prompts and challenges. When you implement flows with an AWS SDK in your application back end, you must construct the logic of requests, prompt users for input, and respond to challenges.

As an application administrator, your user characteristics, security requirements, and authorization model help determine how you want to permit users to sign in. Ask yourself the following questions.

- Do I want to permit users to sign in with credentials from other identity providers (IdPs)?
- Is a username and password enough proof of identity?
- Could my authentication requests for username-password authentication be intercepted? Do I want my application to transmit passwords, or to negotiate authentication using hashes and salts?

- Do I want to permit users to skip password entry and receive a one-time password that signs them in?
- Do I want to permit users to sign in with a thumbprint, face, or a hardware security key?
- When do I want to require multi-factor authentication (MFA), if at all?
- Do I want to persist user sessions without re-prompting for credentials?
- Do I want to extend my authorization model beyond the built-in capabilities of Amazon Cognito?

When you have the answers to these questions, you can learn how to activate the relevant features and implement them in the authentication requests that your application makes.

After you set up sign-in flows for a user, you can check their current status for MFA and choice-based authentication factors with requests to the GetUserAuthFactors API operation. This operation requires authorization with the access token of a signed-in user. It returns user authentication factors and MFA settings.

Topics

- Sign-in with third-party IdPs
- Sign-in with persistent passwords
- Sign-in with persistent passwords and secure payload
- Passwordless sign-in with one-time passwords
- Passwordless sign-in with WebAuthn passkeys
- MFA after sign-in
- Refresh tokens
- Custom authentication
- User migration authentication flow

Sign-in with third-party IdPs

Amazon Cognito user pools serve as an intermediate broker of authentication sessions between IdPs like Sign in with Apple, Login with Amazon, and OpenID Connect (OIDC) services. This process is also called *federated sign-in* or *federated authentication*. Federated authentication doesn't make use of any of the authentication flows that you can build into your app client. Instead, you assign configured user pool IdPs to your app client. Federated sign-in happens when users select their IdP in managed login or your application invokes a session with a redirect to their IdP sign-in page.

With federated sign-in, you delegate primary and MFA authentication factors to the user's IdP. Amazon Cognito doesn't add the other advanced flows in this section to a federated user unless you link them to a local user. Unlinked federated users have usernames, but they are a store of mapped attribute data that's not typically used for sign-in independent of the browser-based flow.

Implementation resources

- User pool sign-in with third party identity providers

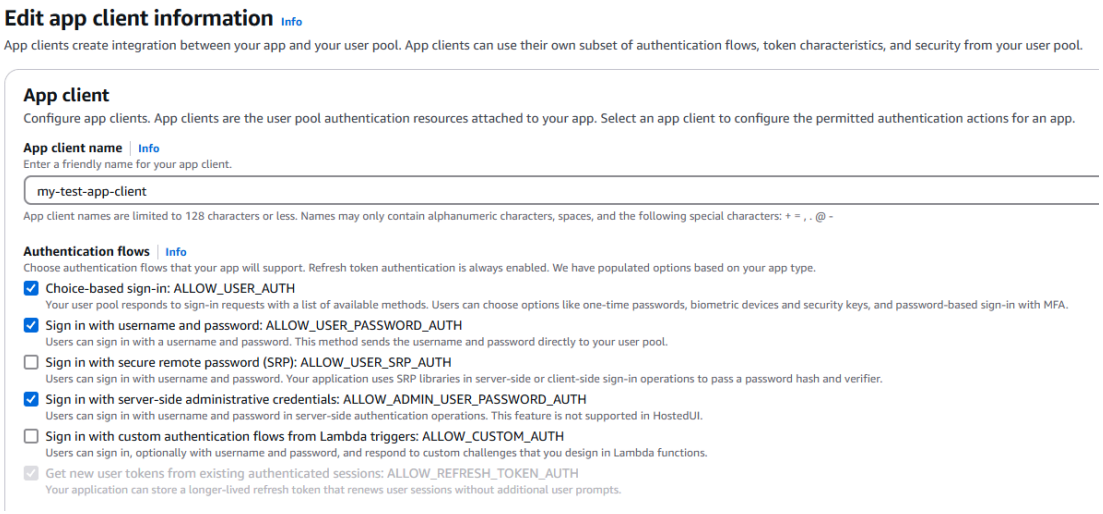
Sign-in with persistent passwords

In Amazon Cognito user pools, every user has a username. This might be a phone number, an email address, or a chosen or administrator-provided identifier. Users of this type can sign in with their username and their password, and optionally provide MFA. User pools can perform username-password sign-in with public or IAM-authorized API operations and SDK methods. Your application can directly send the password to your user pool for authentication. Your user pool responds with additional challenges or the JSON web tokens (JWTs) that are the result of successful authentication.

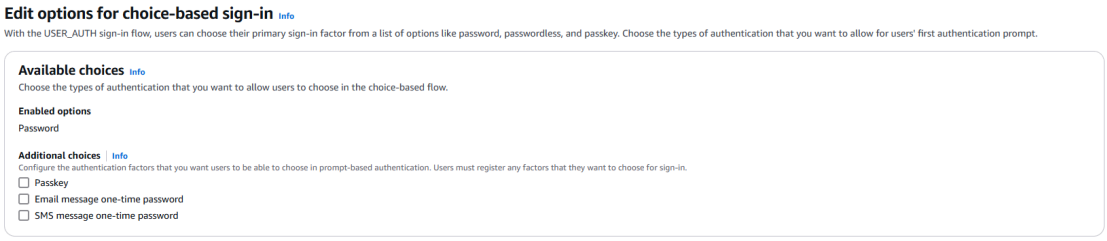
Activate password sign-in

To activate client-based authentication with username and password, configure your app client to permit it. In the Amazon Cognito console, navigate to the **App clients** menu under **Applications** in your user pool configuration. To permit plain-password sign-in for a client-side mobile or native app, edit an app client and choose **Sign in with username and password: ALLOW_USER_PASSWORD_AUTH** under **Authentication flows**. To permit plain-password sign-in for a server-side app, edit an app client and choose **Sign in with server-side administrative credentials: ALLOW_ADMIN_USER_PASSWORD_AUTH**.

To activate choice-based authentication with username and password, configure your app client to permit it. Edit your app client and choose **Choice-based sign-in: ALLOW_USER_AUTH**.



To verify that password authentication is available in choice-based authentication flows, navigate to the **Sign-in menu** and review the section under **Options for choice-based sign-in**. You can sign in with plain-password authentication if **Password** is visible under **Available choices**. The **Password** option includes the plain and SRP username-password authentication variants.



Configure `ExplicitAuthFlows` with your preferred username-and-password authentication options in a CreateUserPoolClient or UpdateUserPoolClient request.

```
"ExplicitAuthFlows": [
   "ALLOW_USER_PASSWORD_AUTH",
   "ALLOW_ADMIN_USER_PASSWORD_AUTH",
   "ALLOW_USER_AUTH"
]
```

In a CreateUserPool or UpdateUserPool request, configure `Policies` with the choice-based authentication flows that you want to support. The `PASSWORD` value in `AllowedFirstAuthFactors` includes both the plain-password and SRP authentication flow options.

```
"Policies": {
```

```
   "SignInPolicy": {
      "AllowedFirstAuthFactors": [
         "PASSWORD",
         "EMAIL_OTP",
         "WEB_AUTHN"
      ]
   }
}
```

Choice-based sign-in with a password

To sign a user in to an application with username-password authentication, configure the body of your AdminInitiateAuth or InitiateAuth request as follows. This sign-in request succeeds or continues to the next challenge if the current user is eligible for username-password authentication. Otherwise, it responds with a list of available primary-factor authentication challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "PREFERRED_CHALLENGE" : "PASSWORD",
      "PASSWORD" : "[User's password]"
   },
   "ClientId": "1example23456789"
}
```

You can also omit the PREFERRED_CHALLENGE value and receive a response that contains a list of eligible sign-in factors for the user.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser"
   },
   "ClientId": "1example23456789"
}
```

If you didn't submit a preferred challenge or the submitted user isn't eligible for their preferred challenge, Amazon Cognito returns a list of options in `AvailableChallenges`.

When `AvailableChallenges` includes a `ChallengeName` of `PASSWORD`, you can continue authentication with a RespondToAuthChallenge or AdminRespondToAuthChallenge challenge response in the format that follows. You must pass a `Session` parameter that associates the challenge response with the API response to your initial sign-in request. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
   "ChallengeName": "PASSWORD",
   "ChallengeResponses": {
      "USERNAME" : "testuser",
      "PASSWORD" : "[User's Password]"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response"
}
```

Amazon Cognito responds to eligible and successful preferred-challenge requests and `PASSWORD` challenge responses with tokens or an additional required challenge like multi-factor authentication (MFA).

Client-based sign-in with a password

To sign a user in to a client-side app with username-password authentication, configure the body of your InitiateAuth request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
   "AuthFlow": "USER_PASSWORD_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "PASSWORD" : "[User's password]"
   },
   "ClientId": "1example23456789"
}
```

To sign a user in to a server-side app with username-password authentication, configure the body of your AdminInitiateAuth request as follows. Your application must sign this request with AWS credentials. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
```

```
   "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "PASSWORD" : "[User's password]"
   },
   "ClientId": "1example23456789"
}
```

Amazon Cognito responds to successful requests with tokens or an additional required challenge like multi-factor authentication (MFA).

Sign-in with persistent passwords and secure payload

Another form of the username-password sign-in methods in user pools is with the Secure Remote Password (SRP) protocol. This option sends proof of knowledge of a password—a password hash and a salt—that your user pool can verify. With no readable secret information in the request to Amazon Cognito, your application is the only entity that processes the passwords that users enter. SRP authentication involves mathematical calculations that are best done by an existing component that you can import in your SDK. The low-level AWS SDKs expose the `InitiateAuth` and `RespondToAuthChallenge` API operations but don't compute the SRP values for you. Higher-level libraries that implement SRP—such as the Amplify libraries—perform these calculations. If your SDK doesn't provide an SRP implementation, you must calculate the SRP values yourself before you call these operations. SRP is typically implemented in client-side applications like mobile apps. For more information about the protocol, see The Stanford SRP Homepage. Wikipedia also has resources and examples. A variety of public libraries are available to perform the SRP calculations for your authentication flows.

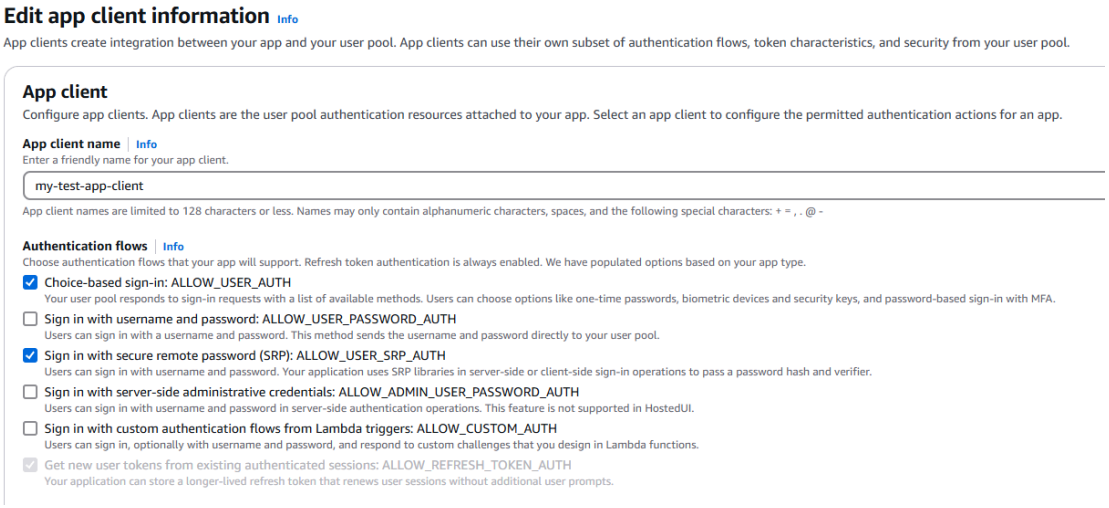
The initiate-challenge-respond sequence of Amazon Cognito authentication validates users and their passwords with SRP. You must configure your user pool and app client to support SRP authentication, then implement the logic of sign-in requests and challenge responses in your application. Your SRP libraries can generate the random numbers and calculated values that demonstrate to your user pool that you are in possession of a user's password. Your application fills in these calculated values to the JSON-formatted `AuthParameters` and `ChallengeParameters` fields in the Amazon Cognito user pools API operations and SDK methods for authentication.

Activate SRP sign-in

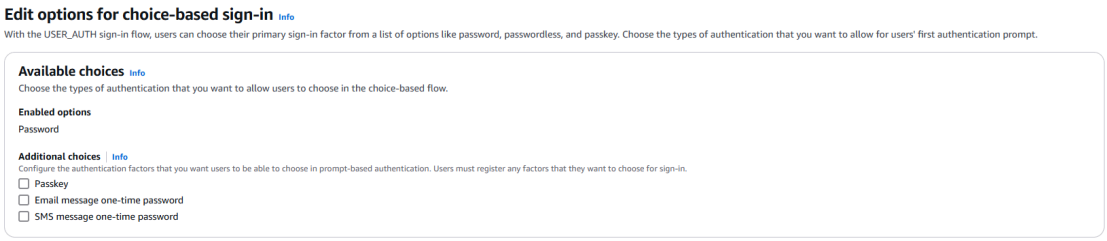
To activate client-based authentication with username and SRP, configure your app client to permit it. In the Amazon Cognito console, navigate to the **App clients** menu under

Applications in your user pool configuration. To permit SRP sign-in for a client-side mobile or native app, edit an app client and choose **Sign in with secure remote password (SRP): ALLOW_USER_SRP_AUTH** under **Authentication flows**.

To activate choice-based authentication with username and SRP, edit your app client and choose **Choice-based sign-in: ALLOW_USER_AUTH**.



To verify that SRP authentication is available in your choice-based authentication flows, navigate to the **Sign-in menu** and review the section under **Options for choice-based sign-in**. You can sign in with SRP authentication if **Password** is visible under **Available choices**. The **Password** option includes the plaintext and SRP username-password authentication variants.



Configure `ExplicitAuthFlows` with your preferred username-and-password authentication options in a CreateUserPoolClient or UpdateUserPoolClient request.

```
"ExplicitAuthFlows": [
   "ALLOW_USER_SRP_AUTH",
   "ALLOW_USER_AUTH"
]
```

In a CreateUserPool or UpdateUserPool request, configure `Policies` with the choice-based authentication flows that you want to support. The `PASSWORD` value in

`AllowedFirstAuthFactors` includes both the plaintext-password and SRP authentication flow options.

```
"Policies": {
   "SignInPolicy": {
      "AllowedFirstAuthFactors": [
         "PASSWORD",
         "EMAIL_OTP",
         "WEB_AUTHN"
      ]
   }
}
```

Choice-based sign-in with SRP

To sign a user in to an application with username-password authentication with SRP, configure the body of your AdminInitiateAuth or InitiateAuth request as follows. This sign-in request succeeds or continues to the next challenge if the current user is eligible for username-password authentication. Otherwise, it responds with a list of available primary-factor authentication challenges. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "PREFERRED_CHALLENGE" : "PASSWORD_SRP",
      "SRP_A" : "[g^a % N]"
   },
   "ClientId": "1example23456789"
}
```

You can also omit the `PREFERRED_CHALLENGE` value and receive a response that contains a list of eligible sign-in factors for the user.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser"
   },
   "ClientId": "1example23456789"
```

```
}
```

If you didn't submit a preferred challenge or the submitted user isn't eligible for their preferred challenge, Amazon Cognito returns a list of options in `AvailableChallenges`. When `AvailableChallenges` includes a `ChallengeName` of `PASSWORD_SRP`, you can continue authentication with a RespondToAuthChallenge or AdminRespondToAuthChallenge challenge response in the format that follows. You must pass a `Session` parameter that associates the challenge response with the API response to your initial sign-in request. This set of parameters is the minimum required for sign-in. Additional parameters are available.

```
{
   "ChallengeName": "PASSWORD_SRP",
   "ChallengeResponses": {
      "USERNAME" : "testuser",
      "SRP_A" : "[g^a % N]"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response"
}
```

Amazon Cognito responds to eligible preferred-challenge requests and `PASSWORD_SRP` challenge responses with a `PASSWORD_VERIFIER` challenge. Your client must complete SRP calculations and respond to the challenge in a RespondToAuthChallenge or AdminRespondToAuthChallenge request.

```
{
   "ChallengeName": "PASSWORD_VERIFIER",
   "ChallengeResponses": {
      "PASSWORD_CLAIM_SIGNATURE" : "string",
      "PASSWORD_CLAIM_SECRET_BLOCK" : "string",
      "TIMESTAMP" : "string"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response]"
}
```

On a successful `PASSWORD_VERIFIER` challenge response, Amazon Cognito issues tokens or another required challenge like multi-factor authentication (MFA).

Client-based sign-in with SRP

SRP authentication is more common to client-side authentication than to server-side. However, you can use SRP authentication with InitiateAuth and AdminInitiateAuth. To sign a user in to an application, configure the body of your `InitiateAuth` or `AdminInitiateAuth` request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

The client generates SRP_A from a generator modulo N *g* raised to the power of a secret random integer *a*.

```
{
   "AuthFlow": "USER_SRP_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "SRP_A" : "[g^a % N]"
   },
   "ClientId": "1example23456789"
}
```

Amazon Cognito responds with a `PASSWORD_VERIFIER` challenge. Your client must complete SRP calculations and respond to the challenge in a RespondToAuthChallenge or AdminRespondToAuthChallenge request.

```
{
   "ChallengeName": "PASSWORD_VERIFIER",
   "ChallengeResponses": {
      "PASSWORD_CLAIM_SIGNATURE" : "string",
      "PASSWORD_CLAIM_SECRET_BLOCK" : "string",
      "TIMESTAMP" : "string"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response]"
}
```

On a successful `PASSWORD_VERIFIER` challenge response, Amazon Cognito issues tokens or another required challenge like multi-factor authentication (MFA).

Passwordless sign-in with one-time passwords

Passwords can be lost or stolen. You might want to verify only that your users have access to a verified email address, phone number, or authenticator app. The solution to this is *passwordless* sign-in. Your application can prompt users to enter their username, email address, or phone number. Amazon Cognito then generates a one-time password (OTP), a code that they must confirm. A successful code completes authentication.

One-time password (OTP) authentication flows aren't compatible with required multi-factor authentication (MFA) in your user pool. Passkey authentication with user verification can satisfy MFA requirements when you set `FactorConfiguration` to `MULTI_FACTOR_WITH_USER_VERIFICATION`. If MFA is optional in your user pool, users who have activated MFA can't sign in with an OTP first factor. Users who don't have an MFA preference in an MFA-optional user pool can sign in with passwordless factors. For more information, see Things to know about user pool MFA.

When a user correctly enters a code they received in an SMS or email message as part of passwordless authentication, in addition to authenticating the user, your user pool marks the user's unverified email address or phone number attribute as verified. The user status also changed from `UNCONFIRMED` to `CONFIRMED`, regardless of whether you configured your user pool to automatically verify email addresses or phone numbers.

New options with passwordless sign-in

When you activate passwordless authentication in your user pool, it changes how some user flows work.

1. Users can sign up without a password and choose a passwordless factor when they sign in. You can also create users without passwords as an administrator.
2. Users who you import with a CSV file can sign in immediately with a passwordless factor. They aren't required to set a password before sign-in.
3. Users who don't have a password can submit ChangePassword API requests without the `PreviousPassword` parameter.

Automatic sign-in with OTPs

Users who sign up and confirm their user accounts with email or SMS message OTPs can automatically sign in with the passwordless factor that matches their confirmation message. In

the managed login UI, users who confirm their accounts and are eligible for OTP sign-in with the confirmation-code delivery method automatically proceed through to their first sign-in after they provide the confirmation code. In your custom-built application with an AWS SDK, pass the following parameters to an InitiateAuth or AdminInitiateAuth operation.

- The `Session` parameter from the ConfirmSignUp API response as the `Session` request parameter.
- An AuthFlow of `USER_AUTH`.

You can pass a PREFERRED_CHALLENGE of `EMAIL_OTP` or `SMS_OTP`, but it's not required. The `Session` parameter provides proof of authentication and Amazon Cognito ignores the `AuthParameters` when you pass a valid session code.

The sign-in operation returns the response that indicates successful authentication, AuthenticationResult, with no additional challenges if the following conditions are true.

- The `Session` code is valid and not expired.
- The user is eligible for the OTP authentication method.

Activate passwordless sign-in

Console

To activate passwordless sign-in, configure your user pool to permit primary sign-in with one or more passwordless types, then configure your app client to permit the `USER_AUTH` flow. In the Amazon Cognito console, navigate to the **Sign-in** menu under **Authentication** in your user pool configuration. Edit **Options for choice-based sign-in** and choose **Email message one-time password** or **SMS message one-time password**. You can activate both options. Save your changes.

Navigate to the **App clients** menu and choose an app client or create a new one. Select **Edit** and choose **Select an authentication type at sign-in: ALLOW_USER_AUTH**.

API/SDK

In the user pools API, configure `SignInPolicy` with the appropriate passwordless options in a CreateUserPool or UpdateUserPool request.

```
"SignInPolicy": {
```

```
    "AllowedFirstAuthFactors": [
        "EMAIL_OTP",
        "SMS_OTP"
    ]
}
```

Configure your app client `ExplicitAuthFlows` with the required option in a CreateUserPoolClient or UpdateUserPoolClient request.

```
"ExplicitAuthFlows": [
   "ALLOW_USER_AUTH"
]
```

Sign in with passwordless

Passwordless sign-in doesn't have a client-based `AuthFlow` that you can specify in InitiateAuth and AdminInitiateAuth. OTP authentication is only available in the choice-based `AuthFlow` of `USER_AUTH`, where you can request a preferred sign-in option or choose the passwordless option from a user's AvailableChallenges. To sign a user in to an application, configure the body of your `InitiateAuth` or `AdminInitiateAuth` request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

In this example, we don't know which way the user wants to sign in. If we add a `PREFERRED_CHALLENGE` parameter and the preferred challenge is available to the user, Amazon Cognito responds with that challenge.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser"
   },
   "ClientId": "1example23456789"
}
```

You can instead add `"PREFERRED_CHALLENGE": "EMAIL_OTP"` or `"PREFERRED_CHALLENGE": "SMS_OTP"` to `AuthParameters` in this example. If the user is eligible for that preferred method, your user pool immediately sends a code to the user's email address or phone number and returns `"ChallengeName": "EMAIL_OTP"` or `"ChallengeName": "SMS_OTP"`.

If you don't specify a preferred challenge, Amazon Cognito responds with an `AvailableChallenges` parameter.

```
{
   "AvailableChallenges": [
      "EMAIL_OTP",
      "SMS_OTP",
      "PASSWORD"
    ],
   "Session": "[Session ID]"
}
```

This user is eligible for passwordless sign-in with email message OTP, SMS message OTP, and username-password. Your application can prompt the user for their selection, or make a selection based on internal logic. It then proceeds with a RespondToAuthChallenge or AdminRespondToAuthChallenge request that selects the challenge. Suppose the user wants to complete passwordless authentication with an email-message OTP.

```
{
   "ChallengeName": "SELECT_CHALLENGE",
   "ChallengeResponses": {
      "USERNAME" : "testuser",
      "ANSWER" : "EMAIL_OTP"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response]"
}
```

Amazon Cognito responds with an `EMAIL_OTP` challenge and sends a code to your user's verified email address. Your application then must respond again to this challenge.

This would also be the next challenge response if you requested `EMAIL_OTP` as a `PREFERRED_CHALLENGE`.

```
{
   "ChallengeName": "EMAIL_OTP",
   "ChallengeResponses": {
      "USERNAME" : "testuser",
      "EMAIL_OTP_CODE" : "123456"
   },
   "ClientId": "1example23456789",
```

```
    "Session": "[Session ID from the previous response]"
}
```

Passwordless sign-in with WebAuthn passkeys

Passkeys are secure and impose a relatively low effort level on users. Passkey sign-in makes use of *authenticators*, external devices that users can authenticate with. Regular passwords expose users to vulnerabilities like phishing, password guessing, and credential theft. With passkeys, your application can benefit from advanced security measures on mobile phones and other devices attached to or built in to information systems. A common passkey sign-in workflow starts with a call to your device that invokes your password or *credentials* manager, for example the iOS keychain or the Google Chrome password manager. The on-device credentials manager prompts them to select a passkey and authorize it with an existing credential or device-unlock mechanism. Modern phones have face scanners, fingerprint scanners, unlock patterns and other mechanisms, some that simultaneously satisfy the *something you know* and *something you have* principles of strong authentication. In the case of passkey authentication with biometrics, passkeys represent a *something you are*.

You might want to replace passwords with the thumbprint, face, or security-key authentication. This is *passkey* or *WebAuthn* authentication. It's common for application developers to permit users to enroll a biometric device after they first sign in with a password. With Amazon Cognito user pools, your application can configure this sign-in option for users. Passkey authentication can satisfy multi-factor authentication (MFA) requirements when your user pool has `FactorConfiguration` set to `MULTI_FACTOR_WITH_USER_VERIFICATION`. In this configuration, passkey authentication with user verification counts as multi-factor authentication.

One-time password (OTP) authentication flows aren't compatible with required multi-factor authentication (MFA) in your user pool. Passkey authentication with user verification can satisfy MFA requirements when you set `FactorConfiguration` to `MULTI_FACTOR_WITH_USER_VERIFICATION`. If MFA is optional in your user pool, users who have activated MFA can't sign in with an OTP first factor. Users who don't have an MFA preference in an MFA-optional user pool can sign in with passwordless factors. For more information, see Things to know about user pool MFA.

What are passkeys?

Passkeys simplify the user experience by eliminating the need to remember complex passwords or enter OTPs. Passkeys are based on WebAuthn and CTAP2 standards drafted by the World

Wide Web Consortium (W3C) and FIDO (Fast Identity Online) Alliance. Browsers and platforms implement these standards, provide APIs for web or mobile applications to start a passkey registration or authentication process, and also UI for user to select and interact with a passkey *authenticator*.

When a user registers an authenticator with a website or an app, the authenticator creates a public-private key pair. WebAuthn browsers and platforms submit the public key to the application back end of the website or app. The authenticator keeps the private key, key IDs, and metadata about the user and application. When the user wants to authenticate in the registered application with their registered authenticator, the application generates a random challenge. The response to this challenge is the digital signature of the challenge generated with the private key of the authenticator for that application and user, and relevant metadata. The browser or application platform receives the digital signature and passes it to the application back end. The application then validates the signature with the stored public key.

> **Note**
>
> Your application doesn't receive any authentication secrets that users provide to their authenticator, nor does it receive information about the private key.

The following are some examples and capabilities of authenticators currently on the market. An authenticator might meet any or all of these categories.

- Some authenticators perform *user verification* with factors like a PIN, biometric input with a face or fingerprint, or a passcode before granting access, ensuring that only the legitimate user can authorize actions. Other authenticators don't have any user verification capabilities, and some can skip user verification when an application doesn't require it.
- Some authenticators, for example YubiKey hardware tokens, are portable. They communicate with devices through USB, Bluetooth or NFC connections. Some authenticators are local and bound to a platform, for example Windows Hello on a PC or Face ID on an iPhone. A device-bound authenticator can be carried by user if small enough, like a mobile device. Sometimes users can connect their hardware authenticator with many different platforms with wireless communication. For example, users in desktop browsers can use their smart phone as a passkey authenticator when they scan a QR code.
- Some platform-bound passkeys sync to the cloud so that they can be used from multiple locations. For example, Face ID passkeys on iPhones sync passkey metadata with users' Apple

accounts in their iCloud Keychain. These passkeys grant seamless authentication across Apple devices, instead of requiring that users register each device independently. Software-based authenticator apps like 1Password, Dashlane, and Bitwarden sync passkeys across all platforms where the user has installed the app.

In WebAuthn terminology, websites and apps are *relying parties*. Each passkey is associated with a specific relying party ID, a unified identifier that represents the websites or apps that accept passkey authentication.. Developers must carefully select their relying party ID to have the right scope of authentication. A typical reliying party ID is the root domain name of a webserver. A passkey with this relying party ID specification can authenticate for that domain and subdomains. Browsers and platforms deny passkey authentication when the URL of the website a user want to access doesn't match the relying party ID. Similarly, for mobile apps, a passkey can only be used if the app path is present in the `.well-known` association files that the application makes available at the path indicated by the relying party ID.

Passkeys are *discoverable*. They can be automatically recognized and used by a browser or platform without requiring the user to input a username. When a user visits a website or app that supports passkey authentication, they can select from a list of passkeys that the browser or platform already knows, or they can scan a QR code.

How does Amazon Cognito implement passkey authentication?

Passkeys are an opt-in feature that's available in all feature plans except for **Lite**. It is only available in the choice-based authentication flow. With managed login, Amazon Cognito handles the logic of passkey authentication. You can also use the Amazon Cognito user pools API in AWS SDKs to do passkey authentication in your application back end.

Amazon Cognito recognizes passkeys created using either of two asymmetric cryptographic algorithms, ES256(-7) and RS256(-257). Most authenticators support both algorithms. By default, users can set up any type of authenticators, for example hardware tokens, mobile smart phones, and software authenticator apps. Amazon Cognito doesn't currently support attestation enforcement.

In your user pool, you can configure user verification to be preferred or required. This setting defaults to preferred in API requests that don't provide a value, and preferred is selected by default in the Amazon Cognito console. When you set user verification to preferred, users can set up authenticators that don't have the user verification capability, and registration and

authentication operations can succeed without user verification. To mandate user verification in passkey registration and authentication, change this setting to required.

The relying party (RP) ID that you set in your passkey configuration is an important decision. When you don't specify otherwise and your domain branding version is managed login, your user pool defaults to expecting the name of your custom domain as the RP ID. If you don't have a custom domain and don't specify otherwise, your user pool defaults to an RP ID of your prefix domain. You can also configure your RP ID to be any domain name not in the public suffix list (PSL). Your RP ID entry applies to passkey registration and authentication in managed login and in SDK authentication. Passkey is only functional in mobile applications with Amazon Cognito can locate a `.well-known` association file with your RP ID as the domain. As a best practice, determine and set the value of your relying party ID before your website or app is publicly available. If you change your RP ID, your users must register again with the new RP ID.

Each user can register up to 20 passkeys. They can only register a passkey after they have signed in to your user pool at least once. Managed login removes significant effort from passkey registration. When you enable passkey authentication for a user pool and app client, your user pool with a managed login domain reminds end users to register a passkey after they sign up for a new user account. You can also invoke users' browsers at any time to direct them to a managed login page for passkey registration. Users must provide a username before Amazon Cognito can initiate passkey authentication. Managed login handles this automatically. The sign-in page prompts for a username, validates that the user has at least one passkey registered, and then prompts for passkey sign-in. Similarly, SDK-based applications must prompt for a username and provide it in the authentication request.

When you set up user pool authentication with passkeys and you have a custom domain and a prefix domain, the RP ID defaults to the fully-qualified domain name (FQDN) of your custom domain. To set a prefix domain as the RP ID in the Amazon Cognito console, delete your custom domain or enter the FQDN of the prefix domain as a **Third-party domain**.

Activate passkey sign-in

Console

To activate sign-in with passkeys, configure your user pool to permit primary sign-in with one or more passwordless types, then configure your app client to permit the USER_AUTH flow. In the Amazon Cognito console, navigate to the **Sign-in** menu under **Authentication** in your user pool configuration. Edit **Options for choice-based sign-in** and add **Passkey** to the list of **Available choices**.

Navigate to the **Authentication methods** menu and edit **Passkey**.

- **User verification** is the setting for whether your user pool requires passkey devices that perform additional checks that the current user is authorized for a passkey. To encourage users to configure a device with user verification, but not require it, select **Preferred**. To only support devices with user verification, select **Required**. For more information, see User verification at w3.org.
- The **Domain for relying party ID** is the identifier that your application will pass in users' passkey registration requests. It sets the target of the trust relationship with the issuer of users' passkeys. Your relying party ID can be: the domain of your user pool if

  Cognito domain

  The Amazon Cognito prefix domain of your user pool.

  Custom domain

  The custom domain of your user pool.

  Third-party domain

  The domain for applications that don't use the user pools managed login pages. This setting is typically associated with user pools that don't have a domain and perform authentication with an AWS SDK and the user pools API in the backend.

Navigate to the **App clients** menu and choose an app client or create a new one. Select **Edit** and under **Authentication flows**, choose **Select an authentication type at sign-in: ALLOW_USER_AUTH**.

API/SDK

In the user pools API, configure `SignInPolicy` with the appropriate passkey options in a CreateUserPool or UpdateUserPool request. The `WEB_AUTHN` option for passkey authentication must be accompanied by at least one other option. Passkey registration requires an existing authentication session.

```
"SignInPolicy": {
    "AllowedFirstAuthFactors": [
        "PASSWORD",
        "WEB_AUTHN"
    ]
```

```
}
```

Configure your user-verification preference and RP ID in the `WebAuthnConfiguration` parameter of a SetUserPoolMfaConfig request. The `RelyingPartyId`, the intended target of passkey authentication outcomes, can be your user pool prefix or custom domain, or a domain of your own choosing.

```
"WebAuthnConfiguration": {
   "RelyingPartyId": "example.auth.us-east-1.amazoncognito.com",
   "UserVerification": "preferred",
   "FactorConfiguration": "SINGLE_FACTOR"
}
```

Configure your app client `ExplicitAuthFlows` with the required option in a CreateUserPoolClient or UpdateUserPoolClient request.

```
"ExplicitAuthFlows": [
   "ALLOW_USER_AUTH"
]
```

Register a passkey (managed login)

Managed login handles user registration of passkeys. When passkey authentication is active in your user pool, Amazon Cognito prompts users to set up a passkey when the register for a new user account.

Amazon Cognito doesn't prompt users to set up a passkey when they have already signed up and not set up a passkey, or if you created their account as an administrator. Users in this state must sign in with another factor like a password or passwordless OTP before they can register a passkey.

To register a passkey

1. Direct the user to your sign-in page.

   ```
   https://auth.example.com/oauth2/authorize/?
   client_id=1example23456789&response_type=code&scope=email+openid
   +phone&redirect_uri=https%3A%2F%2Fwww.example.com
   ```

2. Process the authentication result from the user. In this example, Amazon Cognito redirects them to `www.example.com` with an authorization code that your application exchanges for tokens.
3. Direct the user to your register-passkey page. The user will have a browser cookie that maintains their signed-in session. The passkey URL takes `client_id` and `redirect_uri` parameters. Amazon Cognito only permits authenticated users to access this page. Sign in your user with a password, email OTP, or SMS OTP and then invoke a URL that matches the following pattern.

   You can also add other Authorize endpoint parameters to this request, like `response_type` and `scope`.

   ```
   https://auth.example.com/passkeys/add?
   client_id=1example23456789&redirect_uri=https%3A%2F%2Fwww.example.com
   ```

Register a passkey (SDK)

You register passkey credentials with metadata in a PublicKeyCreationOptions object. You can generate this object with the credentials of a signed-in user and present them in an API request to their passkey issuer. The issuer will return a RegistrationResponseJSON object that confirms passkey registration.

To start the process of passkey registration, sign in a user with an existing sign-in option. Authorize the token-authorized StartWebAuthnRegistration API request with the current user's access token. The following is the body of an example `GetWebAuthnRegistrationOptions` request.

```
{
   "AccessToken": "eyJra456defEXAMPLE"
}
```

The response from your user pool contains the `PublicKeyCreationOptions` object. Present this object in an API request to the user's issuer. It provides information like the public key and relying party ID. The issuer will respond with a `RegistrationResponseJSON` object.

Present the registration response in a CompleteWebAuthnRegistration API request, again authorized with the user's access token. When your user pool responds with an HTTP 200 response with an empty body, your user's passkey is registered.

Sign in with a passkey

Passwordless sign-in doesn't have an `AuthFlow` that you can specify in InitiateAuth and AdminInitiateAuth. Instead, you must declare an `AuthFlow` of `USER_AUTH` and request a sign-in option or choose your passwordless option from the response from your user pool. To sign a user in to an application, configure the body of your `InitiateAuth` or `AdminInitiateAuth` request as follows. This set of parameters is the minimum required for sign-in. Additional parameters are available.

In this example, we know that the user wants to sign in with a passkey, and we add a `PREFERRED_CHALLENGE` parameter.

```
{
   "AuthFlow": "USER_AUTH",
   "AuthParameters": {
      "USERNAME" : "testuser",
      "PREFERRED_CHALLENGE" : "WEB_AUTHN"
   },
   "ClientId": "1example23456789"
}
```

Amazon Cognito responds with a `WEB_AUTHN` challenge. Your application must respond to this challenge. Initiate a sign-in request with the user's passkey provider. It will return an AuthenticationResponseJSON object.

```
{
   "ChallengeName": "WEB_AUTHN",
   "ChallengeResponses": {
      "USERNAME" : "testuser",
      "CREDENTIAL" : "{AuthenticationResponseJSON}"
   },
   "ClientId": "1example23456789",
   "Session": "[Session ID from the previous response]"
}
```

MFA after sign-in

You can set up users who complete sign-in with a username-password flow to be prompted for additional verification with a one-time password from an email message, SMS message, or

code-generating application. MFA is distinct from passwordless sign-in with one-time passwords. However, passkeys with user verification can satisfy MFA requirements as first-factor when you configure `FactorConfiguration` as `MULTI_FACTOR_WITH_USER_VERIFICATION` in your user pool `WebAuthnConfiguration`. Passkeys cannot be used as a second factor to password sign-in. For password-based flows, MFA in user pools is a challenge-response model where a user first demonstrates they know the password, then demonstrates that they have access to their registered second-factor device.

Implementation resources

- Adding MFA to a user pool

Refresh tokens

When you want to keep users signed in without re-entering their credentials, *refresh tokens* are the tool that your application has to persist a user's session. Applications can present refresh tokens to your user pool and exchange them for new ID and access tokens. With token refresh, you can ensure that a signed-in user is still active, get updated attribute information, and update access-control entitlements without user intervention.

Implementation resources

- Refresh tokens

Custom authentication

You might want to configure a method of authentication for your users that isn't listed here. You can do that with *custom authentication* with Lambda triggers. In a sequence of Lambda functions, Amazon Cognito issues a challenge, asks a question that users must answer, checks the answer for accuracy, then determines if another challenge should be issued. The questions and answers can include security questions, requests to a CAPTCHA service, requests to an external MFA service API, or all of these in sequence.

Implementation resources

- Custom authentication challenge Lambda triggers

Custom authentication flow

Amazon Cognito user pools also make it possible to use custom authentication flows, which can help you create a challenge/response-based authentication model using AWS Lambda triggers.

The custom authentication flow makes possible customized challenge and response cycles to meet different requirements. The flow starts with a call to the `InitiateAuth` API operation that indicates the type of authentication to use and provides any initial authentication parameters. Amazon Cognito responds to the `InitiateAuth` call with one of the following types of information:

- A challenge for the user, along with a session and parameters.
- An error if the user fails to authenticate.
- ID, access, and refresh tokens if the supplied parameters in the `InitiateAuth` call are sufficient to sign the user in. (Typically the user or app must first answer a challenge, but your custom code must determine this.)

If Amazon Cognito responds to the `InitiateAuth` call with a challenge, the app gathers more input and calls the `RespondToAuthChallenge` operation. This call provides the challenge responses and passes it back the session. Amazon Cognito responds to the `RespondToAuthChallenge` call similarly to the `InitiateAuth` call. If the user has signed in, Amazon Cognito provides tokens, or if the user isn't signed in, Amazon Cognito provides another challenge, or an error. If Amazon Cognito returns another challenge, the sequence repeats and the app calls `RespondToAuthChallenge` until the user successfully signs in or an error is returned. For more details about the `InitiateAuth` and `RespondToAuthChallenge` API operations, see the API documentation.

Custom authentication flow and challenges

An app can initiate a custom authentication flow by calling `InitiateAuth` with `CUSTOM_AUTH` as the `Authflow`. With a custom authentication flow, three Lambda triggers control challenges and verification of the responses.

- The `DefineAuthChallenge` Lambda trigger uses a session array of previous challenges and responses as input. It then generates the next challenge name and Booleans that indicate whether the user is authenticated and can be granted tokens. This Lambda trigger is a state machine that controls the user's path through the challenges.

- The `CreateAuthChallenge` Lambda trigger takes a challenge name as input and generates the challenge and parameters to evaluate the response. When `DefineAuthChallenge` returns `CUSTOM_CHALLENGE` as the next challenge, the authentication flow calls `CreateAuthChallenge`. The `CreateAuthChallenge` Lambda trigger passes the next type of challenge in the challenge metadata parameter.
- The `VerifyAuthChallengeResponse` Lambda function evaluates the response and returns a Boolean to indicate if the response was valid.

A custom authentication flow can also use a combination of built-in challenges, such as SRP password verification and MFA through SMS. It can use custom challenges such as CAPTCHA or secret questions.

Use SRP password verification in custom authentication flow

If you want to include SRP in a custom authentication flow, you must begin with SRP.

- To initiate SRP password verification in a custom flow, the app calls `InitiateAuth` with `CUSTOM_AUTH` as the `Authflow`. In the `AuthParameters` map, the request from your app includes `SRP_A:` (the SRP A value) and `CHALLENGE_NAME: SRP_A`.
- The `CUSTOM_AUTH` flow invokes the `DefineAuthChallenge` Lambda trigger with an initial session of `challengeName: SRP_A` and `challengeResult: true`. Your Lambda function responds with `challengeName: PASSWORD_VERIFIER`, `issueTokens: false`, and `failAuthentication: false`.
- The app next must call `RespondToAuthChallenge` with `challengeName: PASSWORD_VERIFIER` and the other parameters required for SRP in the `challengeResponses` map.
- If Amazon Cognito verifies the password, `RespondToAuthChallenge` invokes the `DefineAuthChallenge` Lambda trigger with a second session of `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`. At that point, the `DefineAuthChallenge` Lambda trigger responds with `challengeName: CUSTOM_CHALLENGE` to start the custom challenge.
- If MFA is enabled for a user, after Amazon Cognito verifies the password, your user is then challenged to set up or sign in with MFA.

> **Note**
>
> The Amazon Cognito hosted sign-in webpage can't activate Custom authentication challenge Lambda triggers.

For more information about the Lambda triggers, including sample code, see Customizing user pool workflows with Lambda triggers.

User migration authentication flow

A user migration Lambda trigger helps migrate users from a legacy user management system into your user pool. If you choose the `USER_PASSWORD_AUTH` authentication flow, users don't have to reset their passwords during user migration. This flow sends your users' passwords to the service over an encrypted SSL connection during authentication.

When you have migrated all your users, switch flows to the more secure SRP flow. The SRP flow doesn't send any passwords over the network.

To learn more about Lambda triggers, see Customizing user pool workflows with Lambda triggers.

For more information about migrating users with a Lambda trigger, see Importing users with a user migration Lambda trigger.

Authorization models for API and SDK authentication

When you're starting development of your application with user pools authentication, you must decide on the API authorization model that fits your application type. An authorization model is a system for providing authorization to make requests with the authentication components in the Amazon Cognito user pools API and SDK integrations. Amazon Cognito has three authorization models: IAM-authorized, public, and token-authorized.

With IAM-authorized requests, the authorization comes from a signature by a set of AWS IAM credentials in the `Authorization` header of a request. For server-side applications, this practice protects authentication operations with IAM authorization. With public (unauthenticated) authentication requests, no authorization is required. This is suitable for client-side applications distributed to users. With token-authorized operations, typically implemented in combination with public operations, the authorization comes from a session token or an access token included

in the `Authorization` header of the request. Amazon Cognito authentication typically requires that you implement two or more API operations in order, and the API operations you use depend on the characteristics of your application. Public clients, where the application is distributed to users, use public operations, where requests for sign-in don't require authorization. Token-authorized operations continue the session of users in public applications. Server-side clients, where the application logic is hosted on a remote system, protect authentication operations with IAM authorization for sign-in requests. The API operation pairs that follow, and their corresponding SDK methods, map to the available authorization models.

Each public authentication operation has some form of server-side equivalent, for example UpdateUserAttributes and AdminUpdateUserAttributes. While client-side operations are user-initiated and require confirmation, server-side operations assume the change was committed by a user pool administrator and changes take immediate effect. In this example, Amazon Cognito sends a message with a confirmation code to the user, and the user's access token authorizes a VerifyUserAttribute request that submits the code. The server-side application can immediately set the value of any attribute, although special considerations apply for changing the value of email addresses and phone numbers when they're used for sign-in.

To compare API authentication and see a full list of API operations and their authorization models, see Understanding API, OIDC, and managed login pages authentication.

Client-side (public) authentication

The following is a typical sequence of requests in a client-side application

1. The public InitiateAuth operation submits primary credentials like a username and password.
2. The token-authorized RespondToAuthChallenge operation submits a *session* token from the `InitiateAuth` response and the answer to a challenge, for example MFA. Session token authorization indicates requests that are part of not-yet-complete authentication cycles.
3. The token-authorized ConfirmDevice operation submits an *access* token and performs the write operation of adding a remembered device to the user's profile. Access token authorization indicates requests that are for user self-service operations after they have completed authentication.

For more information, see Client-side authentication options and Understanding API, OIDC, and managed login pages authentication.

Server-side authentication

The following is a typical sequence of requests from a server-side operation. Each request has an AWS Signature Version 4 authorization header signed with IAM machine credentials that were issued to the application server.

1. The AdminInitiateAuth operation submits primary credentials like a username and password.
2. AdminRespondToAuthChallenge operation submits the answer to a challenge, for example MFA.
3. The AdminUpdateDeviceStatus operation sets the device key from the `AdminInitiateAuth` response as remembered.

For more information, see Server-side authentication options and Understanding API, OIDC, and managed login pages authentication.

A user authenticates by answering successive challenges until authentication either fails or Amazon Cognito issues tokens to the user. You can repeat these steps with Amazon Cognito, in a process that includes different challenges, to support any custom authentication flow.

Topics

- Server-side authentication options
- Client-side authentication options
- Understanding API, OIDC, and managed login pages authentication
- List of API operations grouped by authorization model

Server-side authentication options

Web applications and other *server-side* applications implement authentication on a remote server that a client loads in a remote-display application like a browser or SSH session. Server-side applications typically have the following characteristics.

- They're built in an application installed on a server in languages like Java, Ruby, or Node.js.
- They connect to user pool app clients that might have a client secret, called *confidential clients*.
- They have access to AWS credentials.

- They invoke managed login for authentication, or use IAM-authorized operations in the user pools API with an AWS SDK.
- They serve internal customers and might serve public customers.

Server-side operations with the user pools API can use passwords, one-time passwords, or passkeys as the primary sign-in factor. For server-side apps, user pool authentication is similar to authentication for client-side apps, except for the following:

- The server-side app makes an AdminInitiateAuth API request. This operation requires AWS credentials with permissions that include `cognito-idp:AdminInitiateAuth` and `cognito-idp:AdminRespondToAuthChallenge`. The operation returns the required challenge or authentication outcome.
- When the application receives a challenge, it makes an AdminRespondToAuthChallenge API request. The `AdminRespondToAuthChallenge` API operation also requires AWS credentials.

For more information about signing Amazon Cognito API requests with AWS credentials, see Signature Version 4 signing process in the *AWS General Reference*.

In the `AdminInitiateAuth` response `ChallengeParameters`, the `USER_ID_FOR_SRP` attribute, if present, contains the user's actual username, not an alias (such as email address or phone number). In your call to `AdminRespondToAuthChallenge`, in the `ChallengeResponses`, you must pass this username in the `USERNAME` parameter.

> **Note**
>
> Because backend admin implementations use the admin authentication flow, the flow doesn't support remembered devices. When you have turned on device tracking, admin authentication succeeds, but any call to refresh the access token fails.

Client-side authentication options

Mobile apps and other *client-side* application types are installed on users' devices and perform the logic of authentication and user interface locally. They typically have the following characteristics.

- They're built in languages like React native, Flutter, and Swift and deploy to user devices.
- They connect to user pool app clients that don't have a client secret, called *public clients*.

- They don't have access to AWS credentials that would authorize IAM-authorized API requests.
- They invoke managed login for authentication, or use public and token-authorized operations in the user pools API with an AWS SDK.
- They serve public customers and permit anyone to sign up and sign in.

Client-side operations with the user pools API can use passwords, one-time passwords, or passkeys as the primary sign-in factor. The following process works for user client-side apps that you create with AWS Amplify or the AWS SDKs.

1. The user enters their username and password into the app.
2. The app calls the `InitiateAuth` operation with the user's username and Secure Remote Password (SRP) details.

   This API operation returns the authentication parameters.

   > **Note**
   >
   > The app generates SRP details with the Amazon Cognito SRP features that are built in to AWS SDKs.

3. The app calls the `RespondToAuthChallenge` operation. If the call succeeds, Amazon Cognito returns the user's tokens, and the authentication flow is complete.

   If Amazon Cognito requires another challenge, the call to `RespondToAuthChallenge` returns no tokens. Instead, the call returns a session.
4. If `RespondToAuthChallenge` returns a session, the app calls `RespondToAuthChallenge` again, this time with the session and the challenge response (for example, MFA code).

Understanding API, OIDC, and managed login pages authentication

Amazon Cognito user pools are a combination of several authentication technologies. They are relying parties to external identity providers (IdPs). They are IdPs to applications that implement authentication with OpenID Connect (OIDC) SDKs. They provide authentication as issuers of JSON web tokens (JWTs) similar to OIDC authentication, but in API methods that are part of AWS SDKs. They can also be secure points of entry to your applications.

When you want to sign up, sign in, and manage users in your user pool, you have two options.

1. Your *managed login pages* and the classic *hosted UI* include the managed login user-interactive endpoints and the federation endpoints that handle IdP and relying-party roles. They make up a package of public webpages that Amazon Cognito activates when you choose a domain for your user pool. For a quick start with the authentication and authorization features of Amazon Cognito user pools, including pages for sign-up, sign-in, password management, and multi-factor authentication (MFA), use the built-in user interface of managed login.

   The other user pool endpoints facilitate authentication with third-party identity providers (IdPs). The services that they perform include the following.

   a. Service-provider callback endpoints for authenticated claims from your IdPs, like `saml2/idpresponse` and `oauth2/idpresponse`. When Amazon Cognito is an intermediate service provider (SP) between your app and your IdP, the callback endpoints represent the service.

   b. Endpoints that provide information about your environment, like `oauth2/userInfo` and `/.well-known/jwks.json`. Your app uses these endpoints when it verifies tokens or retrieves user profile data with OIDC or OAuth 2.0 developer libraries.

2. The Amazon Cognito user pools API is a set of tools for your web or mobile app to authenticate users after it collects sign-in information in your own custom front end. User pools API authentication produces the following JSON web tokens.

   a. An identity token with verifiable attribute claims from your user.

   b. An access token that authorizes your user to create token-authorized API requests to an AWS service endpoint.

   > **Note**
   >
   > By default, access tokens from user pools API authentication only contain the `aws.cognito.signin.user.admin` scope. To generate an access token with additional scopes, for example to authorize a request to a third-party API, request scopes during authentication through your user pool endpoints or add custom scopes in a Pre token generation Lambda trigger. Access token customization adds costs to your AWS bill.

   c. A refresh token that authorizes requests for new ID and access tokens, and refreshes user identity and access-control properties.

You can link a federated user, who would normally sign in through the user pools endpoints, with a user whose profile is *local* to your user pool. A local user exists exclusively in your user pool

directory without federation through an external IdP. If you link their federated identity to a local user in an AdminLinkProviderForUser API request, they can sign in with the user pools API. For more information, see Linking federated users to an existing user profile.

The Amazon Cognito user pools API is dual-purpose.

1. It creates and configures your Amazon Cognito user pools resources. For example, you can create user pools, add AWS Lambda triggers, and configure the user pool domain that hosts your managed login pages.
2. It performs sign-up, sign-in and other user operations for local and linked users.

Example scenario with the Amazon Cognito user pools API

1. Your user selects a "Create an account" button that you created in your app. They enter an email address and password.
2. Your app sends a SignUp API request and creates a new user in your user pool.
3. Your app prompts your user for an email confirmation code. Your users enters the code they received in an email message.
4. Your app sends a ConfirmSignUp API request with the user's confirmation code.
5. Your app prompts your user for their username and password, and they enter their information.
6. Your app sends an InitiateAuth API request and stores an ID token, access token, and refresh token. Your app calls OIDC libraries to manage your user's tokens and maintain a persistent session for that user.

In the Amazon Cognito user pools API, you can't sign in users who federate through an IdP. You must authenticate these users through your user pool endpoints. For more information about the user pool endpoints that include managed login, see User pool endpoints and managed login reference.

Your federated users can start in managed login and select their IdP, or you can skip managed login and send your users directly to your IdP to sign in. When your API request to the Authorize endpoint includes an IdP parameter, Amazon Cognito silently redirects your user to the IdP sign-in page.

Example scenario with managed login pages

1. Your user selects a "Create an account" button that you created in your app.
2. Managed login presents your user with a list of the social identity providers where you have registered developer credentials. Your user chooses Apple.
3. Your app initiates a request to the Authorize endpoint with provider name `SignInWithApple`.
4. Your user's browser opens the Apple authentication page. Your user signs in and chooses to authorize Amazon Cognito to read their profile information.
5. Amazon Cognito confirms the Apple access token and queries your user's Apple profile.
6. Your user presents an Amazon Cognito authorization code to your app.
7. The OIDC library in your application exchanges the authorization code with the Token endpoint and stores an ID token, access token, and refresh token issued by the user pool. Your app uses OIDC libraries to manage your user's tokens and maintain a persistent session for that user.

The user pools API and managed login pages support a variety of scenarios, described throughout this guide. The following sections examine how the user pools API further divides into classes that support your sign-up, sign-in, and resource-management requirements.

List of API operations grouped by authorization model

The Amazon Cognito user pools API, both a resource-management interface and a user-facing authentication and authorization interface, combines the authorization models that follow in its operations. Depending on the API operation, you might have to provide authorization with IAM credentials, an access token, a session token, a client secret, or a combination of these. For many user authentication and authorization operations, you have a choice of authenticated and unauthenticated versions of the request. Unauthenticated operations are best security practice for apps that you distribute to your users, like mobile apps; you don't need to include any secrets in your code.

You can only assign permissions in IAM policies for IAM-authorized management operations and IAM-authorized user operations.

IAM-authorized management operations

IAM-authorized management operations modify and view your user pool and app client configuration, like you would do in the AWS Management Console.

For example, to modify your user pool in an UpdateUserPool API request, you must present AWS credentials and IAM permissions to update the resource.

To authorize these requests in the AWS Command Line Interface (AWS CLI) or an AWS SDK, configure your environment with environment variables or client configuration that adds IAM credentials to your request. For more information, see Accessing AWS using your AWS credentials in the *AWS General Reference*. You can also send requests directly to the service endpoints for the Amazon Cognito user pools API. You must authorize, or *sign*, these requests with AWS credentials that you embed in the header of your request. For more information, see Signing AWS API requests.

IAM-authorized management operations

AddCustomAttributes

CreateGroup

CreateIdentityProvider

CreateResourceServer

CreateUserImportJob

CreateUserPool

CreateUserPoolClient

CreateUserPoolDomain

DeleteGroup

DeleteIdentityProvider

DeleteResourceServer

DeleteUserPool

DeleteUserPoolClient

DeleteUserPoolDomain

IAM-authorized management operations

DescribeIdentityProvider

DescribeResourceServer

DescribeRiskConfiguration

DescribeUserImportJob

DescribeUserPool

DescribeUserPoolClient

DescribeUserPoolDomain

GetCSVHeader

GetGroup

GetIdentityProviderByIdentifier

GetSigningCertificate

GetUICustomization

GetUserPoolMfaConfig

ListGroups

ListIdentityProviders

ListResourceServers

ListTagsForResource

ListUserImportJobs

ListUserPoolClients

ListUserPools

ListUsers

IAM-authorized management operations

ListUsersInGroup

SetRiskConfiguration

SetUICustomization

SetUserPoolMfaConfig

StartUserImportJob

StopUserImportJob

TagResource

UntagResource

UpdateGroup

UpdateIdentityProvider

UpdateResourceServer

UpdateUserPool

UpdateUserPoolClient

UpdateUserPoolDomain

IAM-authorized user operations

IAM-authorized user operations sign up, sign in, manage credentials for, modify, and view your users.

For example, you can have a server-side application tier that backs a web front end. Your server-side app is an OAuth confidential client that you trust with privileged access to your Amazon Cognito resources. To register a user in the app, your server can include AWS credentials in an AdminCreateUser API request. For more information about OAuth client types, see Client Types in *The OAuth 2.0 Authorization Framework*.

To authorize these requests in the AWS CLI or an AWS SDK, configure your server-side app environment with environment variables or client configuration that adds IAM credentials to your request. For more information, see Accessing AWS using your AWS credentials in the *AWS General Reference*. You can also send requests directly to the service endpoints for the Amazon Cognito user pools API. You must authorize, or *sign*, these requests with AWS credentials that you embed in the header of your request. For more information, see Signing AWS API requests.

If your app client has a client secret, you must provide both your IAM credentials and, depending on the operation, either the `SecretHash` parameter or the `SECRET_HASH` value in `AuthParameters`. For more information, see Computing secret hash values.

IAM-authorized user operations

AdminAddUserToGroup

AdminConfirmSignUp

AdminCreateUser

AdminDeleteSoftwareToken

AdminDeleteUser

AdminDeleteUserAttributes

AdminDisableProviderForUser

AdminDisableUser

AdminEnableUser

AdminForgetDevice

AdminGetDevice

AdminGetUser

AdminGetUserAuthFactors

AdminInitiateAuth

IAM-authorized user operations

AdminLinkProviderForUser

AdminListDevices

AdminListGroupsForUser

AdminListUserAuthEvents

AdminRemoveUserFromGroup

AdminResetUserPassword

AdminRespondToAuthChallenge

AdminSetUserMFAPreference

AdminSetUserPassword

AdminSetUserSettings

AdminUpdateAuthEventFeedback

AdminUpdateDeviceStatus

AdminUpdateUserAttributes

AdminUserGlobalSignOut

Unauthenticated user operations

Unauthenticated user operations sign up, sign in, and initiate password resets for your users. Use unauthenticated, or *public*, API operations when you want anyone on the internet to sign up and sign in to your app.

For example, to register a user in your app, you can distribute an OAuth public client that doesn't provide any privileged access to secrets. You can register this user with the unauthenticated API operation SignUp.

To send these requests in a public client that you developed with an AWS SDK, you don't need to configure any credentials. You can also send requests directly to the service endpoints for the Amazon Cognito user pools API with no additional authorization.

If your app client has a client secret, you must provide, depending on the operation, either the `SecretHash` parameter or the `SECRET_HASH` value in `AuthParameters`. For more information, see Computing secret hash values.

Unauthenticated user operations

SignUp

ConfirmSignUp

ResendConfirmationCode

ForgotPassword

ConfirmForgotPassword

InitiateAuth

Token-authorized user operations

Token-authorized user operations sign out, manage credentials for, modify, and view your users after they have signed in or begun the sign-in process. Use token-authorized API operations when you don't want to distribute secrets in your app, and you want to authorize requests with your user's own credentials. If your user has completed sign-in, you must authorize their token-authorized API request with an access token. If your user is in the middle of a sign-in process, you must authorize their token-authorized API request with a session token that Amazon Cognito returned in the response to the previous request.

For example, in a public client, you might want to update a user's profile in a way that restricts the write access to the user's own profile only. To make this update, your client can include the user's access token in a UpdateUserAttributes API request.

To send these requests in a public client that you developed with an AWS SDK, you don't need to configure any credentials. Include an `AccessToken` or `Session` parameter in your request. You can also send requests directly to the service endpoints for the Amazon Cognito user pools API. To

authorize a request to a service endpoint, include the access or session token in the POST body of your request.

To sign an API request for a token-authorized operation, include the access token as an `Authorization` header in your request, in the format `Bearer` *`<Base64-encoded access token>`*.

Token-authorized user operations	AccessToken	Session
RespondToAuthChallenge		✓
ChangePassword	✓	
GetUser	✓	
StartWebAuthnRegistration	✓	
CompleteWebAuthnRegistration	✓	
DeleteWebAuthnCredential	✓	
ListWebAuthnCredentials	✓	
UpdateUserAttributes	✓	
DeleteUserAttributes	✓	
DeleteUser	✓	
ConfirmDevice	✓	
ForgetDevice	✓	
GetDevice	✓	

Token-authorized user operations	AccessToken	Session
ListDevices	✓	
UpdateDeviceStatus	✓	
GetUserAttributeVerificationCode	✓	
VerifyUserAttribute	✓	
SetUserSettings	✓	
SetUserMFAPreference	✓	
GlobalSignOut	✓	
UpdateAuthEventFeedback		✓
AssociateSoftwareToken	✓	✓
VerifySoftwareToken	✓	✓
RevokeToken[1]		
GetTokensFromRefreshToken[1]		

[1] `RevokeToken` and `GetTokensFromRefreshToken` take refresh tokens as the authorization parameter. The refresh token serves as the authorizing token, and as the target resource.

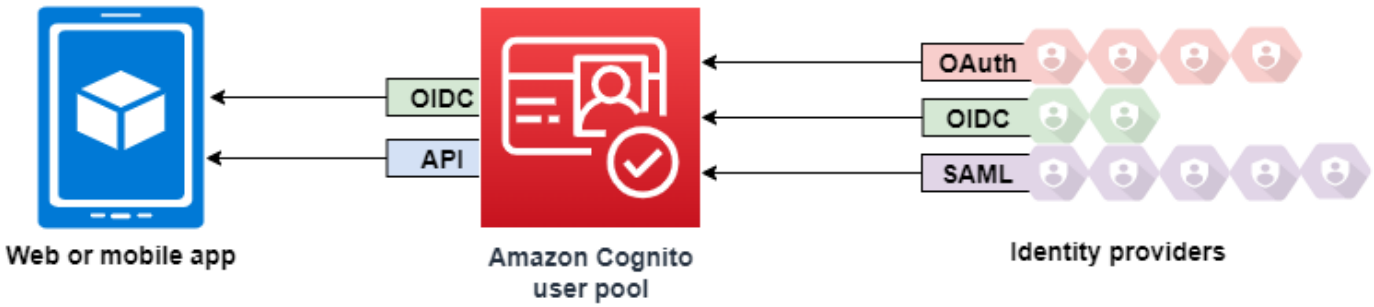
User pool sign-in with third party identity providers

Your app users can either sign in directly through a user pool, or they can federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID

Connect (OIDC) and SAML IdPs. With the built-in hosted web UI, Amazon Cognito provides token handling and management for authenticated users from all IdPs. This way, your backend systems can standardize on one set of user pool tokens.

How federated sign-in works in Amazon Cognito user pools

Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).



Amazon Cognito is a user directory and an OAuth 2.0 identity provider (IdP). When you sign in *local users* to the Amazon Cognito directory, your user pool is an IdP to your app. A local user exists exclusively in your user pool directory without federation through an external IdP.

When you connect Amazon Cognito to social, SAML, or OpenID Connect (OIDC) IdPs, your user pool acts as a bridge between multiple service providers and your app. To your IdP, Amazon Cognito is a service provider (SP). Your IdPs pass an OIDC ID token or a SAML assertion to Amazon Cognito. Amazon Cognito reads the claims about your user in the token or assertion and maps those claims to a new user profile in your user pool directory.

Amazon Cognito then creates a user profile for your federated user in its own directory. Amazon Cognito adds attributes to your user based on the claims from your IdP and, in the case of OIDC and social identity providers, an IdP-operated public `userinfo` endpoint. Your user's attributes change in your user pool when a mapped IdP attribute changes. You can also add more attributes independent of those from the IdP.

After Amazon Cognito creates a profile for your federated user, it changes its function and presents itself as the IdP to your app, which is now the SP. Amazon Cognito is a combination OIDC and OAuth 2.0 IdP. It generates access tokens, ID tokens, and refresh tokens. For more information about tokens, see Understanding user pool JSON web tokens (JWTs).

You must design an app that integrates with Amazon Cognito to authenticate and authorize your users, whether federated or local.

The responsibilities of an app as a service provider with Amazon Cognito

Verify and process the information in the tokens

In most scenarios, Amazon Cognito redirects your authenticated user to an app URL that it appends with an authorization code. Your app exchanges the code for access, ID, and refresh tokens. Then, it must check the validity of the tokens and serve information to your user based on the claims in the tokens.

Respond to authentication events with Amazon Cognito API requests

Your app must integrate with the Amazon Cognito user pools API and the authentication API endpoints. The authentication API signs your user in and out, and manages tokens. The user pools API has a variety of operations that manage your user pool, your users, and the security of your authentication environment. Your app must know what to do next when it receives a response from Amazon Cognito.

Things to know about Amazon Cognito user pools third-party sign-in

- If you want your users to sign in with federated providers, you must choose a domain. This sets up the pages for managed login. For more information, see Using your own domain for managed login.
- You can't sign in federated users with API operations like InitiateAuth and AdminInitiateAuth. When you use the built-in authentication flows of Amazon Cognito, federated users can only sign in with the Login endpoint or the Authorize endpoint. If you use `CUSTOM_AUTH`, your triggers control the outcome. They are responsible for whether the service issues a token to any user (federated or not).
- The Authorize endpoint is a *redirection* endpoint. If you provide an `idp_identifier` or `identity_provider` parameter in your request, it redirects silently to your IdP, bypassing managed login. Otherwise, it redirects to the managed login Login endpoint.
- When managed login redirects a session to a federated IdP, Amazon Cognito includes the `user-agent` header `Amazon/Cognito` in the request.
- Amazon Cognito derives the `username` attribute for a federated user profile from a combination of a fixed identifier and the name of your IdP. To generate a user name that matches your

custom requirements, create a mapping to the `preferred_username` attribute. For more information, see Things to know about mappings.

Example: `MyIDP_bob@example.com`

- Amazon Cognito creates a user group for each OIDC, SAMl, and social IdP that you add to your user pool. The name of the group is in the format `[user pool ID]_[IdP name]`, for example `us-east-1_EXAMPLE_MYSSO` or `us-east-1_EXAMPLE_Google`. Each unique automatically-generated IdP user profile is automatically added to this group. Linked users aren't automatically added to this group, but you can add their profiles to the group in a separate process.
- Amazon Cognito records information about your federated user's identity to an attribute, and a claim in the ID token, called `identities`. This claim contains your user's provider and their unique ID from the provider. You can't change the `identities` attribute in a user profile directly. For more information about how to link a federated user, see Linking federated users to an existing user profile.
- When you update your IdP in an UpdateIdentityProvider API request, your changes can take up to a minute to appear in managed login.
- Amazon Cognito supports up to 20 HTTP redirects between itself and your IdP.
- When your user signs in with managed login, their browser stores an encrypted login-session cookie which records the client and provider they signed in with. If they attempt to sign in again with the same parameters, managed login reuses any *unexpired* existing session, and the user authenticates without providing credentials again. If your user signs in again with a different IdP, including a switch to or from local user pool sign-in, they must provide credentials and generate a new login session.

  You can assign any of your user pool IdPs to any app client, and users can only sign in with an IdP that you assigned to their app client.

Topics

- Configuring identity providers for your user pool
- Using social identity providers with a user pool
- Using SAML identity providers with a user pool
- Using OIDC identity providers with a user pool
- Mapping IdP attributes to profiles and tokens
- Linking federated users to an existing user profile

Configuring identity providers for your user pool

With user pools, you can implement sign-in through a variety of external identity providers (IdPs). This section of the guide has instructions for setting up these identity providers with your user pool in the Amazon Cognito console. Alternatively, you can use the user pools API and an AWS SDK to programmatically add user pool identity providers. For more information, see CreateIdentityProvider.

The supported identity provider options include social providers like Facebook, Google, and Amazon, as well as OpenID Connect (OIDC) and SAML 2.0 providers. Before you get started, set yourself up with administrative credentials for your IdP. For each type of provider, you'll need to register your application, obtain the necessary credentials, and then configure the provider details in your user pool. Your users can then sign up and sign in to your application with their existing accounts from the connected identity providers.

The **Social and external providers** menu under **Authentication** adds and updates user pool IdPs. For more information, see User pool sign-in with third party identity providers.

For an OIDC or OAuth IdP, you can map the ACR values of the IdP to Amazon Cognito authentication levels with the `AcrMapping` field to support step-up authentication for federated users. For more information, see Step-up authentication for federated users.

Topics

- Set up user sign-in with a social IdP
- Set up user sign-in with an OIDC IdP
- Set up user sign-in with a SAML IdP

Set up user sign-in with a social IdP

You can use federation to integrate Amazon Cognito user pools with social identity providers such as Facebook, Google, and Login with Amazon.

To add a social identity provider, you first create a developer account with the identity provider. After you have your developer account, register your app with the identity provider. The identity provider creates an app ID and an app secret for your app, and you configure those values in your Amazon Cognito user pools.

- Google identity platform

- Facebook for developers
- Login with Amazon
- Sign in with Apple

To integrate user sign-in with a social IdP

1. Sign in to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Choose **Add an identity provider**, or choose the **Facebook**, **Google**, **Amazon**, or **Apple** identity provider you have configured, locate **Identity provider information**, and choose **Edit**. For more information about adding a social identity provider, see Using social identity providers with a user pool.
5. Enter your social identity provider's information by completing one of the following steps, based on your choice of IdP:

   Facebook, Google, and Login with Amazon

   Enter the app ID and app secret that you received when you created your client app.

   Sign In with Apple

   Enter the service ID that you provided to Apple, and the team ID, key ID, and private key you received when you created your app client.
6. For **Authorized scopes**, enter the names of the social identity provider scopes that you want to map to user pool attributes. Scopes define which user attributes, such as name and email, that you want to access with your app. When entering scopes, use the following guidelines based on your choice of IdP:

   - **Facebook** — Separate scopes with commas. For example:

     ```
     public_profile, email
     ```

   - **Google, Login with Amazon, and Sign In with Apple** — Separate scopes with spaces. For example:
     - **Google:** `profile email openid`
     - **Login with Amazon:** `profile postal_code`
     - **Sign In with Apple:** `name email`

> **Note**
>
> For Sign In with Apple (console), use the check boxes to choose scopes.

7. Choose **Save changes**.
8. From the **App clients** menu, choose an app client from the list and then select **Edit**. Add the new social identity provider to the app client under **Identity providers**.
9. Choose **Save changes**.

For more information on social IdPs, see Using social identity providers with a user pool.

Set up user sign-in with an OIDC IdP

You can integrate user sign-in with an OpenID Connect (OIDC) identity provider (IdP) such as Salesforce or Ping Identity.

To add an OIDC provider to a user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools** from the navigation menu.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Social and external providers** menu and select **Add an identity provider**.
5. Choose an **OpenID Connect** identity provider.
6. Enter a unique name into **Provider name**.
7. Enter the client ID that you received from your provider into **Client ID**.
8. Enter the client secret that you received from your provider into **Client secret**.
9. Enter **Authorized scopes** for this provider. Scopes define which groups of user attributes (such as `name` and `email`) that your application will request from your provider. Scopes must be separated by spaces, following the OAuth 2.0 specification.

   Your user must consent to provide these attributes to your application.
10. Choose an **Attribute request method** to provide Amazon Cognito with the HTTP method (either GET or POST) that Amazon Cognito uses to fetch the details of the user from the **userInfo** endpoint operated by your provider.

11. Choose a **Setup method** to retrieve OpenID Connect endpoints either by **Auto fill through issuer URL** or **Manual input**. Use **Auto fill through issuer URL** when your provider has a public `.well-known/openid-configuration` endpoint where Amazon Cognito can retrieve the URLs of the `authorization`, `token`, `userInfo`, and `jwks_uri` endpoints.
12. Enter the issuer URL or `authorization`, `token`, `userInfo`, and `jwks_uri` endpoint URLs from your IdP.

    > **Note**
    >
    > You can use only port numbers 443 and 80 with discovery, auto-filled, and manually entered URLs. User logins fail if your OIDC provider uses any nonstandard TCP ports. The issuer URL must start with `https://`, and must not end with a `/` character. For example, Salesforce uses this URL:
    > `https://login.salesforce.com`
    > The `openid-configuration` document associated with your issuer URL must provide HTTPS URLs for the following values: `authorization_endpoint`, `token_endpoint`, `userinfo_endpoint`, and `jwks_uri`. Similarly, when you choose **Manual input**, you can only enter HTTPS URLs.

13. The OIDC claim **sub** is mapped to the user pool attribute **Username** by default. You can map other OIDC claims to user pool attributes. Enter the OIDC claim, and select the corresponding user pool attribute from the drop-down list. For example, the claim **email** is often mapped to the user pool attribute **Email**.
14. Map additional attributes from your identity provider to your user pool. For more information, see Specifying Identity Provider attribute mappings for your user pool.
15. Choose **Create**.
16. From the **App clients** menu, select an app client from the list. To add the new SAML identity provider to the app client, navigate to the **Login pages** tab and select **Edit** on **Managed login pages configuration**.
17. Choose **Save changes**.

For more information on OIDC IdPs, see Using OIDC identity providers with a user pool.

Set up user sign-in with a SAML IdP

You can use federation for Amazon Cognito user pools to integrate with a SAML identity provider (IdP). You supply a metadata document, either by uploading the file or by entering a metadata document endpoint URL. For information about obtaining metadata documents for third-party SAML IdPs, see Configuring your third-party SAML identity provider.

To configure a SAML 2.0 identity provider in your user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Social and external provider** menu and select **Add an identity provider**.
5. Choose a **SAML** identity provider.
6. Enter **Identifiers** separated by commas. An identifier directs Amazon Cognito to check the user sign-in email address, and then direct the user to the provider that corresponds to their domain.
7. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. Configure your SAML 2.0 identity provider to send sign-out responses to the `https://mydomain.auth.us-east-1.amazoncognito.com/saml2/logout` endpoint that Amazon Cognito creates when you configure managed login. The `saml2/logout` endpoint uses POST binding.

   > **Note**
   >
   > If you select this option and your SAML identity provider expects a signed logout request, you also must configure the signing certificate provided by Amazon Cognito with your SAML IdP.
   > The SAML IdP will process the signed logout request and logout your user from the Amazon Cognito session.

8. Choose a **Metadata document source**. If your identity provider offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

> **Note**
>
> If your provider has a public endpoint, we recommend that you enter a metadata document URL, rather than uploading a file. If you use the URL, Amazon Cognito refreshes metadata automatically. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

9. **Map attributes between your SAML provider and your app** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

   For example, when you choose **User pool attribute** `email`, enter the SAML attribute name as it appears in the SAML assertion from your identity provider. Your identity provider might offer sample SAML assertions for reference. Some identity providers use simple names, such as `email`, while others use URL-formatted attribute names similar to:

   ```
   http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
   ```

10. Choose **Create**.

> **Note**
>
> If you see `InvalidParameterException` while creating a SAML IdP with an HTTPS metadata endpoint URL, make sure that the metadata endpoint has SSL correctly set up and that there is a valid SSL certificate associated with it. One example of such an exception would be "Error retrieving metadata from *<metadata endpoint>*".

To set up the SAML IdP to add a signing certificate

- To get the certificate containing the public key that the IdP uses to verify the signed logout request, do the following:
  1. Go to the **Social and external providers** menu of your user pool.
  2. Select your SAML provider,
  3. Choose **View signing certificate.**

For more information on SAML IdPs see Using SAML identity providers with a user pool.

Using social identity providers with a user pool

Your web and mobile app users can sign in through social identity providers (IdP) like Google, Facebook, Amazon, and Apple. You enable managed login, and Amazon Cognito adds a sign-in button for each social IdP to your managed login pages. Amazon Cognito handles the OAuth 2.0 and OpenID Connect exchanges with the provider and issues your app one standard set of user pool tokens, so your backend systems can standardize on user pool tokens regardless of how the user signed in. For more information about the endpoints that Amazon Cognito creates for these exchanges, see the Amazon Cognito user pools Auth API reference.

Each provider has its own developer console, its own values to enter, and its own pitfalls. Choose your provider for step-by-step instructions:

- Add Google as a social identity provider
- Add Sign in with Apple as a social identity provider
- Add Facebook as a social identity provider
- Add Login with Amazon as a social identity provider

> **Note**
>
> Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of federation through Amazon Cognito identity pools (federated identities).

Topics

- Add Google as a social identity provider
- Add Sign in with Apple as a social identity provider
- Add Facebook as a social identity provider
- Add Login with Amazon as a social identity provider

Add Google as a social identity provider

> **Note**
>
> **Last verified against the provider console:** October 2, 2026. Google console steps follow the Google Auth Platform (the sections **Branding**, **Audience**, **Data Access**, and **Clients**). If the Google console differs from these steps, consult the Google Auth Platform documentation.

To let your users sign in with Google, you register an OAuth client in the Google Auth Platform, then add Google as an identity provider (IdP) in your user pool. You must enable managed login first.

Register an OAuth client in the Google Auth Platform

To register a Google OAuth client

1. Sign in to the Google Cloud console and create or select a project.
2. Open **Menu**, then **Google Auth Platform**. If this is your first time, choose **Get started** on the **Overview** page and provide your app name, user support email, audience type (**External**), and contact information.
3. Choose **Branding**. Under **Authorized domains**, add the registrable domains of your user pool, not the full URLs:
   - `amazoncognito.com`, for the default Amazon Cognito domain.
   - The root of your custom domain, for example `example.com`, if you use one.
4. Choose **Data Access**, then **Add or remove scopes**. Add `openid`, `.../auth/userinfo.email`, and `.../auth/userinfo.profile`.
5. Choose **Clients**, then **Create client**. For **Application type**, choose **Web application**.
6. Under **Authorized JavaScript origins**, enter your user pool domain with no path:
   - Default Amazon Cognito domain: `https://<your-prefix>.auth.<region>.amazoncognito.com`
   - Custom domain: `https://auth.example.com`
7. Under **Authorized redirect URIs**, enter the `/oauth2/idpresponse` endpoint of your user pool domain:

- Default Amazon Cognito domain: `https://<your-prefix>.auth.<region>.amazoncognito.com/oauth2/idpresponse`
- Custom domain: `https://auth.example.com/oauth2/idpresponse`

8. Choose **Create**. Securely store the **Client ID** and **Client secret** that Google displays. You enter these values when you add Google to your user pool.

> **Note**
>
> For clients created after June 2025, Google shows the client secret only once. Store it immediately.

Add Google to your user pool

To add Google as an IdP in the AWS Management Console

1. In the Amazon Cognito console, choose your user pool, then **Social and external providers**, then **Add an identity provider**.
2. Choose **Google**. Enter the **Client ID** and **Client secret** from the Google Auth Platform.
3. For **Authorized scopes**, enter `profile email openid`, separated by spaces.
4. Map the Google attributes you want to your user pool attributes. At a minimum, map `email` to `email` and `email_verified` to `email_verified`. If you don't map `email_verified`, Amazon Cognito treats the email as unverified.
5. Choose **Add identity provider**, and enable Google on your app client.

Test Google sign-in

Open your managed login sign-in page and choose **Continue with Google**, or open the authorize endpoint directly:

```
https://<your-prefix>.auth.<region>.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=<app-client-id>&redirect_uri=<your-app-
callback>&identity_provider=Google
```

Checkpoint

The browser completes the Google account chooser and returns through `/oauth2/idpresponse` to your app with an authorization code. In **User management**, a new federated user appears with the Google provider, a populated `email`, and `email_verified` set to `true`. You should see no `redirect_uri_mismatch` error.

Google-specific pitfalls

Unverified consent screen user cap

An app with the **Testing** publishing status is limited to the test users you list. An app that shows the unverified-app screen is capped at 100 new users for the lifetime of the project, and the cap can't be reset. However, if your app requests only the basic sign-in scopes (`openid`, `email`, `profile`), test users don't need to be listed, see no warning, and their authorizations don't expire. Keep to these scopes to avoid the cap, or verify your app in the Verification Center.

Redirect-URI propagation

Changes to authorized origins and redirect URIs can take from five minutes to a few hours to take effect. A mismatch returns `redirect_uri_mismatch`; an unregistered origin returns `origin_mismatch`.

email_verified and account linking

Google asserts `email_verified=true` for Gmail and Google Workspace accounts. Amazon Cognito doesn't automatically merge a Google user with an existing user that has the same email. To link them, implement linking (for example, with a pre sign-up Lambda trigger and `AdminLinkProviderForUser`), and only link on a trusted `email_verified` value.

Add Sign in with Apple as a social identity provider

> **Note**
>
> **Last verified against the provider console:** October 2, 2026. Apple steps use Certificates, Identifiers & Profiles in the Apple Developer account. You must have the Account Holder or Admin role.

Sign in with Apple requires four artifacts: an App ID, a Services ID (your Amazon Cognito client ID), a private key, and your Team ID. You then add Apple as an identity provider (IdP) in your user pool. You must enable managed login first.

Create your Apple identifiers and key

To create the Apple artifacts

1. In **Certificates, Identifiers & Profiles**, create an **App ID** (Identifiers, then **+**, then **App IDs**, then **App**) and enable the **Sign in with Apple** capability.
2. Create a **Services ID** (Identifiers, then **+**, then **Services IDs**). Enter a description and a reverse-domain identifier, for example `com.example.auth.cognito`. This identifier is your Amazon Cognito **Services ID** (client ID).
3. Select the Services ID, enable **Sign in with Apple**, and choose **Configure**. Choose your primary App ID, then enter the web URLs:
   - **Domains and Subdomains** (host only, no scheme): `<your-prefix>.auth.<region>.amazoncognito.com` for the default Amazon Cognito domain, or `auth.example.com` for a custom domain.
   - **Return URLs** (full URL with `https://`): `https://<your-prefix>.auth.<region>.amazoncognito.com/oauth2/idpresponse`, or `https://auth.example.com/oauth2/idpresponse` for a custom domain.
4. Create a private key (Keys, then **+**), enable **Sign in with Apple**, and download the `.p8` file. You can download it only once. Record the **Key ID**.
5. Record your 10-character **Team ID** from the Membership page. You enter the Services ID, Team ID, Key ID, and private key when you add Apple to your user pool.

Add Apple to your user pool

To add Apple as an IdP in the AWS Management Console

1. In the Amazon Cognito console, choose your user pool, then **Social and external providers**, then **Add an identity provider**, then **Sign in with Apple**.
2. Enter the **Services ID**, **Team ID**, **Key ID**, and **private key**. Amazon Cognito uses the private key to generate the client secret for you.
3. For **Authorized scopes**, enter `email name`. Use only `email` if you don't need the user's name.
4. Map `email` to `email` and `sub` to your user name attribute. Apple returns the user's name only on the first sign-in, so capture it then if you map it.
5. Choose **Add identity provider**, and enable Apple on your app client.

Test Sign in with Apple

Open your managed login sign-in page and choose **Continue with Apple**. Test both **Share My Email** and **Hide My Email**.

Checkpoint

The browser returns through `/oauth2/idpresponse` to your app with an authorization code. A user appears in your user pool with the **SignInWithApple** provider and a `sub`. The Share My Email run shows the real email; the Hide My Email run shows a `@privaterelay.appleid.com` address. An `invalid_client` error almost always means the client-secret JWT has expired or the Services ID or Return URL doesn't match `/oauth2/idpresponse` exactly.

Apple-specific pitfalls

The client secret is a JWT that expires

Apple's client secret is not a static string. It's an ES256-signed JWT with a maximum lifetime of six months (15,777,000 seconds), and when it expires, Apple sign-in breaks with `invalid_client` and no warning. In the configuration described here, you give Amazon Cognito the private key (along with the Services ID, Team ID, and Key ID), and Amazon Cognito generates the client-secret JWT from that key—you don't paste a pre-built secret. Keep the private key you registered with Apple valid and current in the IdP configuration; if you revoke or replace the Apple key, update the IdP (for example, with `UpdateIdentityProvider`) with the new key.

Private relay email and account linking

When a user chooses **Hide My Email**, Apple returns a `@privaterelay.appleid.com` address instead of the real one. This address differs from the email the same person uses with other providers, so don't link accounts across identity providers by email when Apple is involved—link by the provider `sub` instead. To send mail to a relay address, register your sending domains under **Sign in with Apple for Email Communication**.

email_verified type and managed accounts

Apple's `email_verified` claim can arrive as the string `"true"` or the boolean `true`—handle both. It's always true for consumer Apple Accounts. For Apple at Work & School (managed) accounts it can be false. Prefer `sub`-based linking and normalize the string value before any merge logic.

Add Facebook as a social identity provider

> **Note**
>
> **Last verified against the provider console:** October 2, 2026. Facebook steps use the Meta for Developers console, which organizes app setup around **Use cases**. The OAuth redirect settings are under **Use cases**, **Customize**, **Settings**.

To let your users sign in with Facebook, you create a Meta app with the Facebook Login use case, then add Facebook as an identity provider (IdP) in your user pool. You must enable managed login first.

Create a Meta app for Facebook Login

To create and configure a Meta app

1. Sign in to Meta for Developers, choose **My Apps**, then **Create App**.
2. For the use case, choose **Authenticate and request data from users with Facebook Login**. Finish creating the app. New apps start in **Development** mode.
3. From **App settings**, then **Basic**, record the **App ID** and **App secret**. Set **App Domains** to your user pool domain host (`<your-prefix>.auth.<region>.amazoncognito.com` or `auth.example.com`), and add the **Website** platform with your managed login sign-in URL as the **Site URL**.
4. Go to **Use cases**, choose your Facebook Login use case, then **Customize**. Confirm the `public_profile` permission (required) and add `email`.
5. In the use case **Settings**, set **Valid OAuth Redirect URIs** to the `/oauth2/idpresponse` endpoint of your user pool domain:

   - Default Amazon Cognito domain: `https://<your-prefix>.auth.<region>.amazoncognito.com/oauth2/idpresponse`
   - Custom domain: `https://auth.example.com/oauth2/idpresponse`

Add Facebook to your user pool

To add Facebook as an IdP in the AWS Management Console

1. In the Amazon Cognito console, choose your user pool, then **Social and external providers**, then **Add an identity provider**, then **Facebook**.
2. Enter the **App ID** and **App secret**, and choose a Facebook API version. Choose the latest available version, because each version has a discontinuation date and scopes can vary between versions.
3. For **Authorized scopes**, enter `public_profile,email`, separated by commas.
4. Map `email` to `email` and `name` to `name`.
5. Choose **Add identity provider**, and enable Facebook on your app client.

Test Facebook sign-in

While the app is in **Development** mode, add your own Facebook account as an app role (Administrator, Developer, or Tester). Then open your managed login sign-in page and choose **Continue with Facebook**.

Checkpoint

Facebook returns through `/oauth2/idpresponse` to your app with an authorization code, and a new federated user appears in your user pool with `email` and `name` populated. Then switch the app to **Live** mode and confirm a non-role account can sign in. A non-role account that sees "app not active" or "in development mode" means the app is still in Development mode or lacks Advanced Access.

Facebook-specific pitfalls

Development mode and app review

> A new app is in **Development** mode and only app-role users (Administrator, Developer, Tester) can sign in. To let the general public sign in, switch the app to **Live** mode, complete Business Verification, and obtain Advanced Access for `email` (and increased access for `public_profile`). First-time testers often hit "app not active" because they test with a non-role account.

No email_verified claim

Facebook does not return an `email_verified` claim, so there is nothing to map, and Amazon Cognito treats the Facebook email as unverified. Don't assume a verified email from Facebook, and link accounts by email only through deliberate, trusted logic rather than automatically.

Keep scopes small

Sign-in completion drops sharply past about four permissions. Keep the request to `public_profile` and `email`. Amazon Cognito federation is also not compatible with Facebook Limited Login on iOS.

Add Login with Amazon as a social identity provider

> **Note**
>
> **Last verified against the provider console:** October 2, 2026. Login with Amazon steps use the Amazon Developer console (not the AWS console).

To let your users sign in with their Amazon account, you register a Login with Amazon security profile, then add Login with Amazon as an identity provider (IdP) in your user pool. You must enable managed login first.

Register a Login with Amazon security profile

To create and configure a security profile

1. Sign in to the Amazon Developer console, or go to **Apps and Services**, then **Login with Amazon**.
2. Choose **Create a New Security Profile**. Enter a **Name** (shown on the consent screen), a **Description**, and a **Consent Privacy Notice URL**. Choose **Save**.
3. Reveal the **Client ID** and **Client Secret** for the security profile. You enter these when you add Login with Amazon to your user pool.
4. Open **Web Settings** for the profile and choose **Edit**. Enter the values below, then **Save**.
   - **Allowed Origins** (HTTPS, no path): `https://<your-prefix>.auth.<region>.amazoncognito.com` or `https://auth.example.com`.

- **Allowed Return URLs** (the `/oauth2/idpresponse` endpoint): `https://<your-prefix>.auth.<region>.amazoncognito.com/oauth2/idpresponse` or `https://auth.example.com/oauth2/idpresponse`. Amazon Cognito uses the redirect flow, so the return URL is the value it depends on.

Add Login with Amazon to your user pool

To add Login with Amazon as an IdP in the AWS Management Console

1. In the Amazon Cognito console, choose your user pool, then **Social and external providers**, then **Add an identity provider**, then **Login with Amazon**.
2. Enter the **Client ID** and **Client Secret** from the security profile.
3. For **Authorized scopes**, enter `profile`. This returns the user's name, email, and user ID. (Add `postal_code` only if you need the ZIP or postal code.)
4. Map `email` to `email`, `name` to `name`, and `user_id` to your user name attribute.
5. Choose **Add identity provider**, and enable Login with Amazon on your app client.

Test Login with Amazon sign-in

Open your managed login sign-in page and choose **Continue with Amazon**, then authenticate and accept the consent screen.

Checkpoint

The browser returns through `/oauth2/idpresponse` to your app with an authorization code, and the user appears in your user pool as a federated user whose user name is prefixed `LoginWithAmazon_`. The ID token contains the mapped `email` and `name`. Note that `email_verified` reads `false` unless you set it with a trigger (see pitfalls).

Login with Amazon pitfalls

No email_verified claim

The `profile` scope returns only `email`, `name`, and `user_id`—there is no `email_verified` claim to map. Mapped emails are unverified by default, so Login with Amazon users land with `email_verified` set to `false`.

Effect on account linking

Because the email is unverified, Amazon Cognito won't automatically merge a Login with Amazon user into an existing email-verified user. To link or mark the email verified, use a pre sign-up Lambda trigger (and `AdminLinkProviderForUser`), and only when your trust model justifies trusting the Amazon-asserted email.

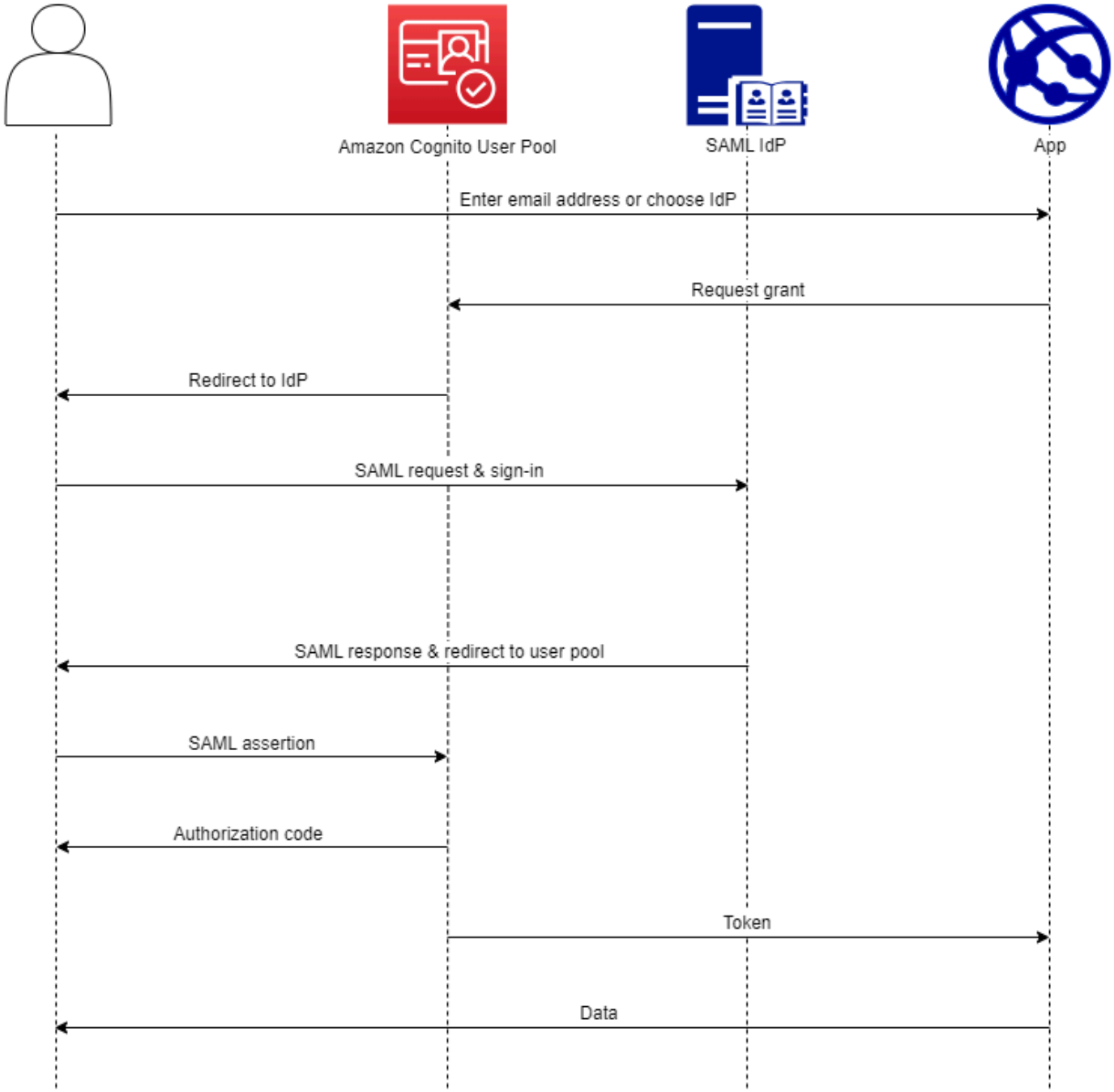
Mapped attributes must be writable

Any custom attribute you map must be mutable and writable by the app client, or sign-in fails or silently skips the attribute. Standard `email` and `name` are fine.

Using SAML identity providers with a user pool

You can choose to have your web and mobile app users sign in through a SAML identity provider (IdP) like Microsoft Active Directory Federation Services (ADFS), or Shibboleth. You must choose a SAML IdP which supports the SAML 2.0 standard.

With managed login, Amazon Cognito authenticates local and third-party IdP users and issues JSON web tokens (JWTs). With the tokens that Amazon Cognito issues, you can consolidate multiple identity sources into a universal OpenID Connect (OIDC) standard across all of your apps. Amazon Cognito can process SAML assertions from your third-party providers into that SSO standard. You can create and manage a SAML IdP in the AWS Management Console, through the AWS CLI, or with the Amazon Cognito user pools API. To create your first SAML IdP in the AWS Management Console, see Adding and managing SAML identity providers in a user pool.



Note

Federation with sign-in through a third-party IdP is a feature of Amazon Cognito user pools. Amazon Cognito identity pools, sometimes called Amazon Cognito federated identities, are an implementation of federation that you must set up separately in

each identity pool. A user pool can be a third-party IdP to an identity pool. For more information, see Amazon Cognito identity pools.

Quick reference for IdP configuration

You must configure your SAML IdP to accept request and send responses to your user pool. The documentation for your SAML IdP will contain information about how to add your user pool as a relying party or application for your SAML 2.0 IdP. The documentation that follows provides the values that you must provide for the SP entity ID and assertion consumer service (ACS) URL.

User pool SAML values quick reference

SP entity ID

```
urn:amazon:cognito:sp:us-east-1_EXAMPLE
```

ACS URL

```
https://Your user pool domain/saml2/idpresponse
```

You must configure your user pool to support your identity provider. The high-level steps to add an external SAML IdP are as follows.

1. Download SAML metadata from your IdP, or retrieve the URL to your metadata endpoint. See Configuring your third-party SAML identity provider.
2. Add a new IdP to your user pool. Upload the SAML metadata or provide the metadata URL. See Adding and managing SAML identity providers in a user pool.
3. Assign the IdP to your app clients. See Application-specific settings with app clients.

Topics

- Things to know about SAML IdPs in Amazon Cognito user pools
- Case sensitivity of SAML user names
- Configuring your third-party SAML identity provider
- Adding and managing SAML identity providers in a user pool

- SAML session initiation in Amazon Cognito user pools
- Signing out SAML users with single sign-out
- SAML signing and encryption
- SAML identity provider names and identifiers

Things to know about SAML IdPs in Amazon Cognito user pools

Implementation of a SAML 2.0 IdP comes with some requirements and restrictions. Refer to this section when you're implementing your IdP. You'll also find information that's useful for troubleshooting errors during SAML federation with a user pool.

Amazon Cognito processes SAML assertions for you

Amazon Cognito user pools support SAML 2.0 federation with POST-binding endpoints. This eliminates the need for your app to retrieve or parse SAML assertion responses, because the user pool directly receives the SAML response from your IdP through a user agent. Your user pool acts as a service provider (SP) on behalf of your application. Amazon Cognito supports SP-initiated and IdP-initiated single sign-on (SSO) as described in sections 5.1.2 and 5.1.4 of the SAML V2.0 Technical Overview.

Provide a valid IdP signing certificate

The signing certificate in your SAML provider metadata must not be expired when you configure the SAML IdP in your user pool.

User pools support multiple signing certificates

When your SAML IdP includes more than one signing certificate in SAML metadata, at sign-in your user pool determines that the SAML assertion is valid if it matches any certificate in the SAML metadata. Each signing certificate must be no longer than 4,096 characters in length.

Maintain the relay state parameter

Amazon Cognito and your SAML IdP maintain session information with a `relayState` parameter.

1. Amazon Cognito supports `relayState` values greater than 80 bytes. While SAML specifications state that the `relayState` value "must not exceed 80 bytes in length", current industry practice often deviates from this behavior. As a consequence, rejecting `relayState` values greater than 80 bytes will break many standard SAML provider integrations.

2. The `relayState` token is an opaque reference to state information maintained by Amazon Cognito. Amazon Cognito doesn't guarantee the contents of the `relayState` parameter. Don't parse its contents such that your app depends on the result. For more information, see the SAML 2.0 specification.

Identify the ACS endpoint

Your SAML identity provider requires that you set an assertion consumer endpoint. Your IdP redirects your users to this endpoint with their SAML assertion. Configure the following endpoint in your user pool domain for SAML 2.0 POST binding in your SAML identity provider.

```
https://Your user pool domain/saml2/idpresponse
With an Amazon Cognito domain:
https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse
With a custom domain:
https://auth.example.com/saml2/idpresponse
```

See Configuring a user pool domain for more information about user pool domains.

No replayed assertions

You can't repeat, or *replay*, a SAML assertion to your Amazon Cognito `saml2/idpresponse` endpoint. A replayed SAML assertion has an assertion ID that duplicates the ID of an earlier IdP response.

User pool ID is SP entity ID

You must provide your IdP with your user pool ID in the service provider (SP) `urn`, also called the *audience URI* or *SP entity ID*. The audience URI for your user pool has the following format.

```
urn:amazon:cognito:sp:us-east-1_EXAMPLE
```

You can find your user pool ID under **User pool overview** in the Amazon Cognito console.

Map all required attributes

Configure your SAML IdP to provide values for any attributes that you set as required in your user pool. For example, `email` is a common required attribute for user pools. Before your users can sign in, your SAML IdP assertions must include a claim that you map to the **User pool attribute** `email`. For more information about attribute mapping, see Mapping IdP attributes to profiles and tokens.

Assertion format has specific requirements

Your SAML IdP must include the following claims in the SAML assertion.

- A `NameID` claim. Amazon Cognito associates a SAML assertion with the destination user by `NameID`. If `NameID` changes, Amazon Cognito considers the assertion to be for a new user. The attribute that you set to `NameID` in your IdP configuration must have a persistent value. To assign SAML users to a consistent user profile in your user pool, assign your `NameID` claim from an attribute with a value that doesn't change.

  ```
  <saml2:NameID Format="urn:oasis:names:tc:SAML:1.1:nameid-format:persistent">
    carlos
  </saml2:NameID>
  ```

  A `Format` in your IdP `NameID` claim of `urn:oasis:names:tc:SAML:1.1:nameid-format:persistent` indicates that your IdP is passing an unchanging value.
  Amazon Cognito doesn't require this format declaration, and assigns a format of `urn:oasis:names:tc:SAML:1.1:nameid-format:unspecified` if your IdP doesn't specify a format of the `NameID` claim. This behavior complies with section 2.2.2 *Complex Type NameIDType*, of the SAML 2.0 specification.

- An `AudienceRestriction` claim with an `Audience` value that sets your user pool SP entity ID as the target of the response.

  ```
  <saml:AudienceRestriction>
    <saml:Audience> urn:amazon:cognito:sp:us-east-1_EXAMPLE
  </saml:AudienceRestriction>
  ```

- For SP-initiated single sign-on, a `Response` element with an `InResponseTo` value of the original SAML request ID.

  ```
  <saml2p:Response Destination="https://mydomain.auth.us-east-1.amazoncognito.com/
  saml2/idpresponse" ID="id123" InResponseTo="_dd0a3436-bc64-4679-
  a0c2-cb4454f04184" IssueInstant="Date-time stamp" Version="2.0"
   xmlns:saml2p="urn:oasis:names:tc:SAML:2.0:protocol" xmlns:xs="http://
  www.w3.org/2001/XMLSchema">
  ```

  > **Note**
  >
  > IdP-initiated SAML assertions must *not* contain an `InResponseTo` value.

- A `SubjectConfirmationData` element with a `Recipient` value of your user pool `saml2/idpresponse` endpoint and, for SP-initiated SAML, an `InResponseTo` value that matches the original SAML request ID.

```
<saml2:SubjectConfirmationData InResponseTo="_dd0a3436-bc64-4679-a0c2-
cb4454f04184" NotOnOrAfter="Date-time stamp" Recipient="https://mydomain.auth.us-
east-1.amazoncognito.com/saml2/idpresponse"/>
```

SP-initiated sign-in requests

When the Authorize endpoint redirects your user to your IdP sign-in page, Amazon Cognito includes a *SAML request* in a URL parameter of the `HTTP GET` request. A SAML request contains information about your user pool, including your ACS endpoint. You can optionally apply a cryptographic signature to these requests.

Sign requests and encrypt responses

Every user pool with a SAML provider generates an asymmetric key pair and *signing certificate* for a digital signature that Amazon Cognito assigns to SAML requests. Every external SAML IdP that you configure to support encrypted SAML response causes Amazon Cognito to generate a new key pair and *encryption certificate* for that provider. To view and download the certificates with the public key, choose your IdP in the **Social and external providers** menu in the Amazon Cognito console.

To establish trust with SAML requests from your user pool, provide your IdP with a copy of your user pool SAML 2.0 signing certificate. Your IdP might ignore SAML requests that your user pool signed if you don't configure the IdP to accept signed requests.

1. Amazon Cognito applies a digital signature to SAML requests that your user passes to your IdP. Your user pool signs all single logout (SLO) requests, and you can configure your user pool to sign single sign-on (SSO) requests for any SAML external IdP. When you provide a copy of the certificate, your IdP can verify the integrity of your users' SAML requests.
2. Your SAML IdP can encrypt SAML responses with the encryption certificate. When you configure an IdP with SAML encryption, your IdP must only send encrypted responses.

Encode non-alphanumeric characters

Amazon Cognito doesn't accept 4-byte UTF-8 characters like # or # that your IdP passes as an attribute value. You can encode the character to Base64, pass it as text, and then decode it in your app.

In the following example, the attribute claim will not be accepted:

```
<saml2:Attribute Name="Name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:unspecified">
  <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="xsd:string">#</saml2:AttributeValue>
</saml2:Attribute>
```

In contrast to the preceding example, the following attribute claim will be accepted:

```
<saml2:Attribute Name="Name" NameFormat="urn:oasis:names:tc:SAML:2.0:attrname-
format:unspecified">
  <saml2:AttributeValue xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
 xsi:type="xsd:string">8J+YkA==</saml2:AttributeValue>
</saml2:Attribute>
```

Metadata endpoint must have valid transport-layer security

If you see `InvalidParameterException` while creating a SAML IdP with an HTTPS metadata endpoint URL, for example, "Error retrieving metadata from *<metadata endpoint>*," make sure that the metadata endpoint has SSL correctly set up and that there is a valid SSL certificate associated with it. For more information about validating certificates, see What Is An SSL/TLS Certificate?.

Metadata endpoint must be on standard TCP port for HTTP or HTTPS

Amazon Cognito only accepts metadata URLs for SAML providers on the standard TCP ports 80 for HTTP and 443 for HTTPS. As a security best practice, host SAML metadata at a TLS-encrypted URL with the `https://` prefix. Enter metadata URLs in the format *`http://www.example.com/saml2/metadata.xml`* or *`https://www.example.com/saml2/metadata.xml`*. The Amazon Cognito console accepts metadata URLs only with the `https://` prefix. You can also configure IdP metadata with CreateIdentityProvider and UpdateIdentityProvider.

App clients with IdP-initiated SAML can only sign in with SAML

When you activate support for a SAML 2.0 IdP that supports IdP-initiated sign in an app client, you can only add other SAML 2.0 IdPs to that app client. You're prevented from adding the user directory in the user pool *and* all non-SAML external identity providers to an app client configured in this way.

Logout responses must use POST binding

The `/saml2/logout` endpoint accepts `LogoutResponse` as `HTTP POST` requests. User pools don't accept logout responses with `HTTP GET` binding.

Metadata Signing Certificate Rotation

Amazon Cognito caches SAML metadata for up to six hours when you provide metadata with a URL. When performing any metadata signing certificate rotation, configure your metadata source to publish *both* the original and new certificates for at least six hours. When Amazon Cognito refreshes the cache from the metadata URL, it treats each certificate as valid and your SAML IdP can start signing SAML assertions with the new certificate. After this period has elapsed, you can remove the original certificate from the published metadata.

Case sensitivity of SAML user names

When a federated user attempts to sign in, the SAML identity provider (IdP) passes a unique `NameId` to Amazon Cognito in the user's SAML assertion. Amazon Cognito identifies a SAML-federated user by their `NameId` claim. Regardless of the case sensitivity settings of your user pool, Amazon Cognito recognizes a returning federated user from a SAML IdP when they pass their unique and case-sensitive `NameId` claim. If you map an attribute like `email` to `NameId`, and your user changes their email address, they can't sign in to your app.

Map `NameId` in your SAML assertions from an IdP attribute that has values that don't change.

For example, Carlos has a user profile in your case-insensitive user pool from an Active Directory Federation Services (ADFS) SAML assertion that passed a `NameId` value of `Carlos@example.com`. The next time Carlos attempts to sign in, your ADFS IdP passes a `NameId` value of `carlos@example.com`. Because `NameId` must be an exact case match, the sign-in doesn't succeed.

If your users can't log in after their `NameID` changes, delete their user profiles from your user pool. Amazon Cognito will create new user profiles the next time they sign in.

Topics

- Configuring your third-party SAML identity provider
- Adding and managing SAML identity providers in a user pool
- SAML session initiation in Amazon Cognito user pools

- Signing out SAML users with single sign-out
- SAML signing and encryption
- SAML identity provider names and identifiers

Configuring your third-party SAML identity provider

When you want to add a SAML identity provider (IdP) to your user pool, you must make some configuration updates in the management interface of your IdP. This section describes how to format the values that you must provide to your IdP. You can also learn about how to retrieve the static or active-URL metadata document that identifies the IdP and its SAML claims to your user pool.

To configure third-party SAML 2.0 identity provider (IdP) solutions to work with federation for Amazon Cognito user pools, you must configure your SAML IdP to redirect to the following Assertion Consumer Service (ACS) URL: `https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse`. If your user pool has an Amazon Cognito domain, you can find your user pool domain path in the **Domain** menu of your user pool in the Amazon Cognito console.

Some SAML IdPs require that you provide the `urn`, also called the audience URI or SP entity ID, in the form `urn:amazon:cognito:sp:us-east-1_EXAMPLE`. You can find your user pool ID under **User pool overview** in the Amazon Cognito console.

You must also configure your SAML IdP to provide values for any attributes that you designated as *required attributes* in your user pool. Typically, `email` is a required attribute for user pools, in which case the SAML IdP must provide some form of an `email` claim in their SAML assertion, and you must map the claim to the attribute for that provider.

The following configuration information for third-party SAML 2.0 IdP solutions is a good place to start setting up federation with Amazon Cognito user pools. For the most current information, consult your provider's documentation directly.

To sign SAML requests, you must configure your IdP to trust requests signed by your user pool signing certificate. To accept encrypted SAML responses, you must configure your IdP to encrypt *all* SAML responses to your user pool. Your provider will have documentation about configuring these features. For an example from Microsoft, see Configure Microsoft Entra SAML token encryption.

> **Note**
>
> Amazon Cognito only requires your identity provider metadata document. Your provider might also offer customized configuration information for SAML 2.0 federation with IAM or AWS IAM Identity Center. To learn how to set up Amazon Cognito integration, look for general directions for retrieving the metadata document and manage the rest of the configuration in your user pool.

Solution	More information
Microsoft Entra ID	Federation Metadata
Okta	How to Download the IdP Metadata and SAML Signing Certificates for a SAML App Integration
Auth0	Configure Auth0 as SAML Identity Provider
Ping Identity (PingFederate)	Exporting SAML metadata from PingFederate
JumpCloud	SAML Configuration Notes
SecureAuth	SAML application integration

Adding and managing SAML identity providers in a user pool

After you configure your identity provider to work with Amazon Cognito, you can add it to your user pools and app clients. The following procedures demonstrate how to create, modify, and delete SAML providers in an Amazon Cognito user pool.

AWS Management Console

You can use the AWS Management Console to create and delete SAML identity providers (IdPs).

Before you create a SAML IdP, you must have the SAML metadata document that you get from the third-party IdP. For instructions on how to get or generate the required SAML metadata document, see Configuring your third-party SAML identity provider.

To configure a SAML 2.0 IdP in your user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Social and external providers** menu and then select **Add an identity provider**.
5. Choose a **SAML** IdP.
6. Enter a **Provider name**. You can pass this friendly name in an `identity_provider` request parameter to the Authorize endpoint.
7. Enter **Identifiers** separated by commas. An identifier tells Amazon Cognito it should check the email address a user enters when they sign in, and then direct them to the provider that corresponds to their domain.
8. Choose **Add sign-out flow** if you want Amazon Cognito to send signed sign-out requests to your provider when a user logs out. You must configure your SAML 2.0 IdP to send sign-out responses to the `https://`*`mydomain.auth.us-east-1.amazoncognito.com`*`/saml2/logout` endpoint that is created when you configure managed login. The `saml2/logout` endpoint uses POST binding.

   > **Note**
   >
   > If this option is selected and your SAML IdP expects a signed logout request, you must also provide your SAML IdP with the signing certificate from your user pool. The SAML IdP will process the signed logout request and sign out your user from the Amazon Cognito session.

9. Choose your **IdP-initiated SAML sign-in** configuration. As a security best practice, choose **Accept SP-initiated SAML assertions only**. If you have prepared your environment to securely accept unsolicited SAML sign-in sessions, choose **Accept SP-initiated and IdP-initiated SAML assertions**. For more information, see SAML session initiation in Amazon Cognito user pools.
10. Choose a **Metadata document source**. If your IdP offers SAML metadata at a public URL, you can choose **Metadata document URL** and enter that public URL. Otherwise, choose **Upload metadata document** and select a metadata file you downloaded from your provider earlier.

> **Note**
>
> We recommend that you enter a metadata document URL if your provider has a public endpoint instead of uploading a file. Amazon Cognito automatically refreshes metadata from the metadata URL. Typically, metadata refresh happens every 6 hours or before the metadata expires, whichever is earlier.

11. **Map attributes between your SAML provider and your user pool** to map SAML provider attributes to the user profile in your user pool. Include your user pool required attributes in your attribute map.

    For example, when you choose **User pool attribute** `email`, enter the SAML attribute name as it appears in the SAML assertion from your IdP. If your IdP offers sample SAML assertions, you can use these sample assertions to help you to find the name. Some IdPs use simple names, such as `email`, while others use names like the following.

    ```
    http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
    ```

12. Choose **Create**.

API/CLI

Use the following commands to create and manage a SAML identity provider (IdP).

To create an IdP and upload a metadata document

- AWS CLI: `aws cognito-idp create-identity-provider`

  Example with metadata file: `aws cognito-idp create-identity-provider --user-pool-id` *`us-east-1_EXAMPLE`* `--provider-name=`*`SAML_provider_1`* `--provider-type SAML --provider-details file:///details.json --attribute-mapping` *`email`*`=`*`http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`*

  Where `details.json` contains:

  ```
  "ProviderDetails": {
        "MetadataFile": "<SAML metadata XML>",
        "IDPSignout" : "true",
  ```

```
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

> **Note**
>
> If the *<SAML metadata XML>* contains any instances of the character ", you must add \ as an escape character: \".

Example with metadata URL: `aws cognito-idp create-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1 --provider-type SAML --provider-details MetadataURL=https://myidp.example.com/sso/saml/metadata --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: CreateIdentityProvider

To upload a new metadata document for an IdP

- AWS CLI: `aws cognito-idp update-identity-provider`

  Example with metadata file: `aws cognito-idp update-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1 --provider-details file:///details.json --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

  Where `details.json` contains:

```
"ProviderDetails": {
      "MetadataFile": "<SAML metadata XML>",
      "IDPSignout" : "true",
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

> **Note**
>
> If the *<SAML metadata XML>* contains any instances of the character ", you must add \ as an escape character: \".

Example with metadata URL: `aws cognito-idp update-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1 --provider-details MetadataURL=https://myidp.example.com/sso/saml/metadata --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

- AWS API: UpdateIdentityProvider

To get information about a specific IdP

- AWS CLI: `aws cognito-idp describe-identity-provider`

  `aws cognito-idp describe-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1`
- AWS API: DescribeIdentityProvider

To list information about all IdPs

- AWS CLI: `aws cognito-idp list-identity-providers`

  Example: `aws cognito-idp list-identity-providers --user-pool-id us-east-1_EXAMPLE --max-results 3`
- AWS API: ListIdentityProviders

To delete an IdP

- AWS CLI: `aws cognito-idp delete-identity-provider`

  `aws cognito-idp delete-identity-provider --user-pool-id us-east-1_EXAMPLE --provider-name=SAML_provider_1`
- AWS API: DeleteIdentityProvider

To set up the SAML IdP to add a user pool as a relying party

- The user pools service provider URN is: `urn:amazon:cognito:sp:`*`us-east-1_EXAMPLE`*. Amazon Cognito requires an audience restriction value that matches this URN in the SAML response. Configure your IdP to use the following POST binding endpoint for the IdP-to-SP response message.

  ```
  https://mydomain.auth.us-east-1.amazoncognito.com/saml2/idpresponse
  ```

- Your SAML IdP must populate `NameID` and any required attributes for your user pool in the SAML assertion. `NameID` is used for uniquely identifying your SAML federated user in the user pool. Your IdP must pass each user's SAML name ID in a consistent, case-sensitive format. Any variation in the value of a user's name ID creates a new user profile.

To provide a signing certificate to your SAML 2.0 IDP

- To download a copy of the the public key from Amazon Cognito that your IdP can use to validate SAML logout requests, choose the **Social and external providers** menu of your user pool, select your IdP, and under **View signing certificate**, select **Download as .crt**.

You can delete any SAML provider you have set up in your user pool with the Amazon Cognito console.

To delete a SAML provider

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Select the radio button next to the SAML IdPs you wish to delete.
5. When you are prompted to **Delete identity provider**, enter the name of the SAML provider to confirm deletion, and then choose **Delete**.

SAML session initiation in Amazon Cognito user pools

Amazon Cognito supports service provider-initiated (SP-initiated) single sign-on (SSO) and IdP-initiated SSO. As a best security practice, implement SP-initiated SSO in your user pool. Section 5.1.2 of the SAML V2.0 Technical Overview describes SP-initiated SSO. Amazon Cognito is the

identity provider (IdP) to your app. The app is the service provider (SP) that retrieves tokens for authenticated users. However, when you use a third-party IdP to authenticate users, Amazon Cognito is the SP. When your SAML 2.0 users authenticate with an SP-initiated flow, they must always first make a request to Amazon Cognito and redirect to the IdP for authentication.

For some enterprise use cases, access to internal applications starts at a bookmark on a dashboard hosted by the enterprise IdP. When a user selects a bookmark, the IdP generates a SAML response and sends it to the SP to authenticate the user with the application.

You can configure a SAML IdP in your user pool to support IdP-initiated SSO. When you support IdP-initiated authentication, Amazon Cognito can't verify that it has solicited the SAML response that it receives because Amazon Cognito doesn't initiate authentication with a SAML request. In SP-initiated SSO, Amazon Cognito sets state parameters that validate a SAML response against the original request. With SP-initiated sign-in you can also guard against cross-site request forgery (CSRF).

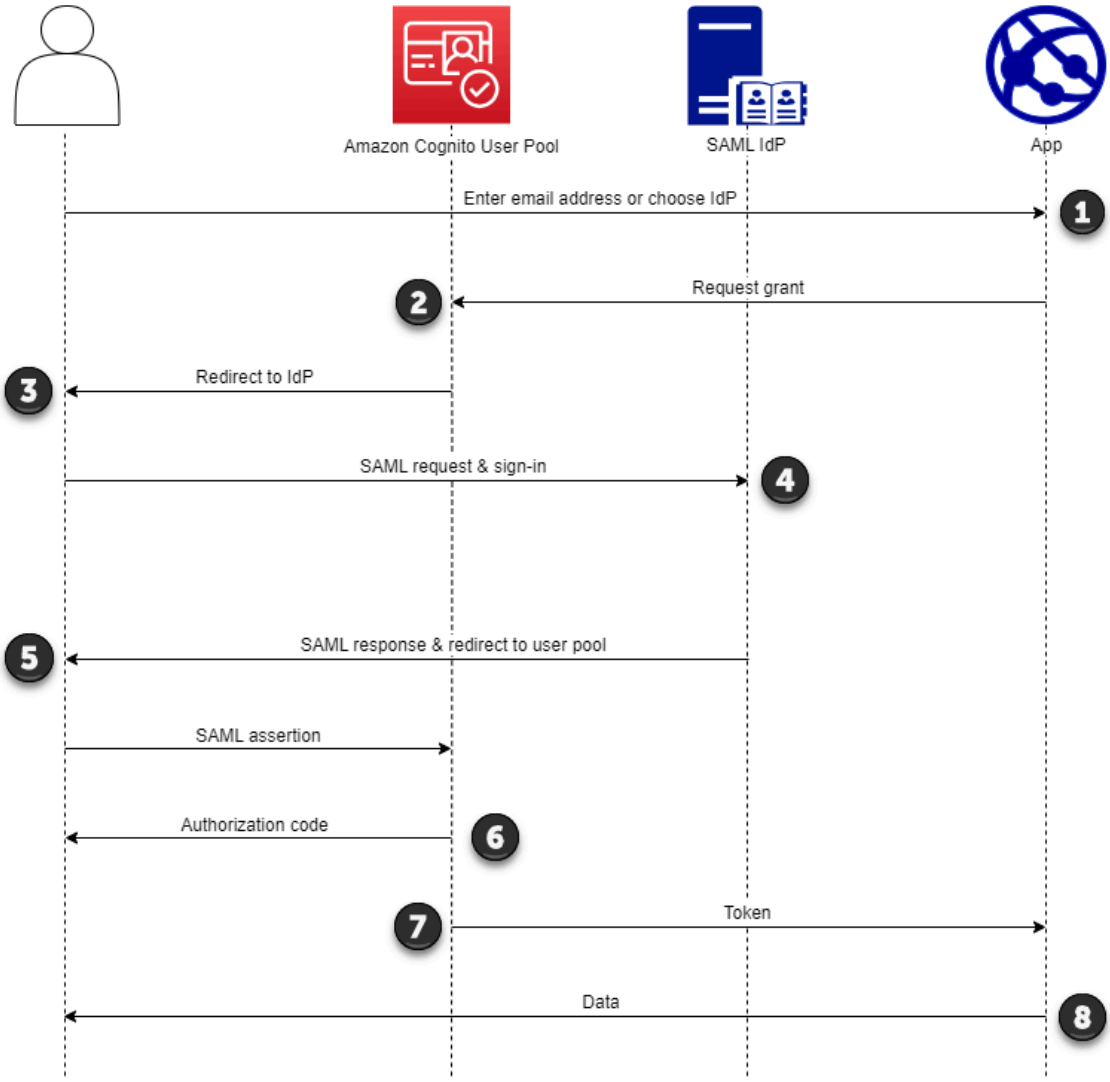
Topics

- Implement SP-initated SAML sign-in
- Implement IdP-initiated SAML sign-in

Implement SP-initated SAML sign-in

As a best practice, implement service-provider-initiated (SP-initiated) sign-in to your user pool. Amazon Cognito initiates your user's session and redirects them to your IdP. With this method, you have the greatest control over who presents sign-in requests. You can also permit IdP-initiated sign-in under certain conditions.

The following process shows how users complete SP-initiated sign in to your user pool through a SAML provider.



1. Your user enters their email address at a sign-in page. To determine your user's redirect to their IdP, you can collect their email address in a custom-built application or invoke managed login in web view.

   You can configure your managed login pages to display a list of IdPs or to prompt for an email address and match it to the identifier of your SAML IdP. To prompt for an email address, edit your managed login branding style and in **Foundation**, locate **Authentication behavior** and under **Provider display**, set **Display style** to **Domain search input**.
2. Your app invokes your user pool redirect endpoint and requests a session with the client ID that corresponds to the app and the IdP ID that corresponds to the user.
3. Amazon Cognito redirects your user to the IdP with a SAML request, optionally signed, in an `AuthnRequest` element.

4. The IdP authenticates the user interactively, or with a remembered session in a browser cookie.
5. The IdP redirects your user to your user pool SAML response endpoint with the optionally-encrypted SAML assertion in their POST payload.

   > **Note**
   >
   > Amazon Cognito cancels sessions that don't receive a response within 5 minutes, and redirects the user to managed login. When your user experiences this outcome, they receive a `Something went wrong` error message.

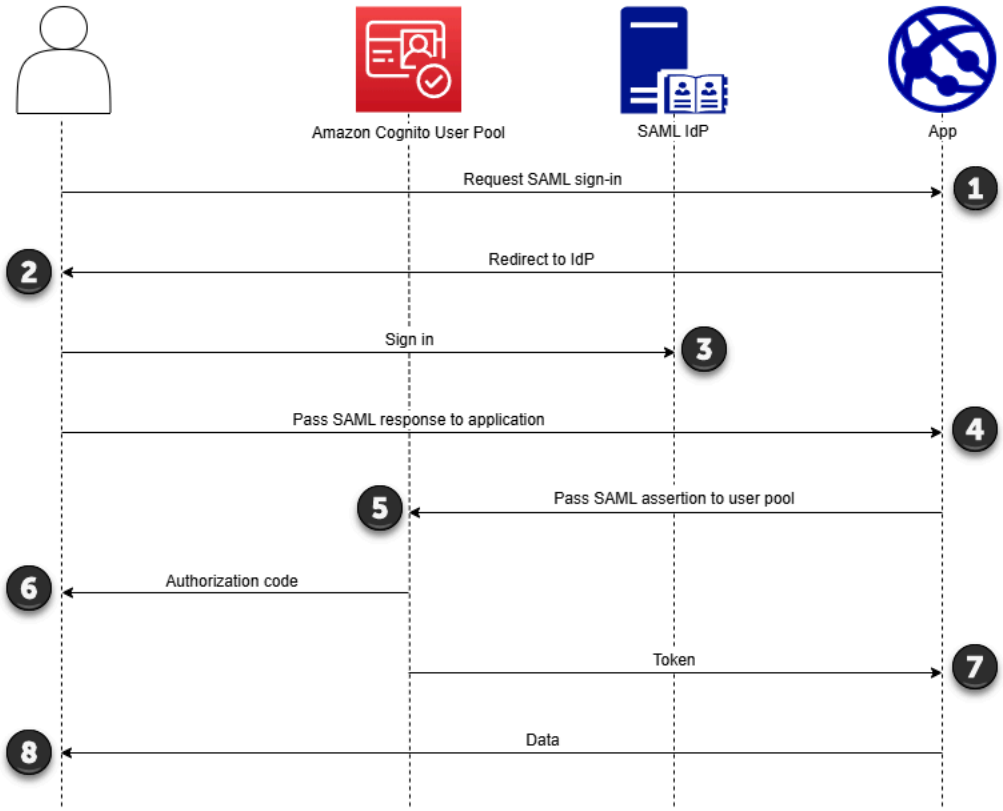
6. After it verifies the SAML assertion and maps user attributes from the claims in the response, Amazon Cognito internally creates or updates the user's profile in the user pool. Typically, your user pool returns an authorization code to your user's browser session.
7. Your user presents their authorization code to your app, which exchanges the code for JSON web tokens (JWTs).
8. Your app accepts and processes your user's ID token as authentication, generates authorized requests to resources with their access token, and stores their refresh token.

When a user authenticates and receives an authorization code grant, the user pool returns ID, access, and refresh tokens. The ID token is a authentication object for OIDC-based identity management. The access token is an authorization object with OAuth 2.0 scopes. The refresh token is an object that generates new ID and access tokens when your user's current tokens have expired. You can configure the duration of users' tokens in your user pool app client.

You can also choose the duration of refresh tokens. After a user's refresh token expires, they must sign in again. If they authenticated through a SAML IdP, your users' session duration is set by the expiration of their tokens, not the expiration of their session with their IdP. Your app must store each user's refresh token and renew their session when it expires. Managed login maintains user sessions in a browser cookie that's valid for 1 hour.

Implement IdP-initiated SAML sign-in

When you configure your identity provider for IdP-initiated SAML 2.0 sign-in, you can present SAML assertions to the `saml2/idpresponse` endpoint in your user pool domain without the need to initiate the session at the Authorize endpoint. A user pool with this configuration accepts IdP-initiated SAML assertions from a user pool external identity provider that the requested app client supports.



1. A user requests SAML sign-in with your application.
2. Your application invokes a browser or redirects the user to the sign-in page for their SAML provider.
3. The IdP authenticates the user interactively, or with a remembered session in a browser cookie.
4. The IdP redirects your user to your application with the SAML assertion, or response, in their POST body.
5. Your application adds the SAML assertion to the POST body of a request to your user pool `saml2/idpresponse` endpoint.
6. Amazon Cognito issues an authorization code to your user.
7. Your user presents their authorization code to your app, which exchanges the code for JSON web tokens (JWTs).
8. Your application accepts and processes your user's ID token as authentication, generates authorized requests to resources with their access token, and stores their refresh token.

The following steps describe the overall process to configure and sign in with an IdP-initiated SAML 2.0 provider.

1. Create or designate a user pool and app client.
2. Create a SAML 2.0 IdP in your user pool.
3. Configure your IdP to support IdP initiation. IdP-initiated SAML introduces security considerations that other SSO providers aren't subject to. Because of this, you can't add non-SAML IdPs, including the user pool itself, to any app client that uses a SAML provider with IdP-initiated sign-in.
4. Associate your IdP-initiated SAML provider with an app client in your user pool.
5. Direct your user to the sign-in page for your SAML IdP and retrieve a SAML assertion.
6. Direct your user to your user pool `saml2/idpresponse` endpoint with their SAML assertion.
7. Receive JSON web tokens (JWTs).

To accept unsolicited SAML assertions in your user pool, you must consider its effect on your app security. Request spoofing and CSRF attempts are likely when you accept IdP-initiated requests. Although your user pool can't verify an IdP-initiated sign-in session, Amazon Cognito validates your request parameters and SAML assertions.

Additionally, your SAML assertion must not contain an `InResponseTo` claim and must have been issued within the previous 6 minutes.

You must submit requests with IdP-initiated SAML to your `/saml2/idpresponse`. For SP-initiated and managed login authorization requests, you must provide parameters that identify your requested app client, scopes, redirect URI, and other details as query string parameters in `HTTP GET` requests. For IdP-initiated SAML assertions, however, the details of your request must be formatted as a `RelayState` parameter in the body of an `HTTP POST` request. The request body must also contain your SAML assertion as a `SAMLResponse` parameter.

The following is an example request and response for an IdP-initiated SAML provider.

```
POST /saml2/idpresponse HTTP/1.1
User-Agent: USER_AGENT
Accept: */*
Host: example.auth.us-east-1.amazoncognito.com
Content-Type: application/x-www-form-urlencoded

SAMLResponse=[Base64-encoded SAML assertion]&RelayState=identity_provider
%3DMySAMLIdP%26client_id%3D1example23456789%26redirect_uri%3Dhttps%3A%2F
%2Fwww.example.com%26response_type%3Dcode%26scope%3Demail%2Bopenid%2Bphone
```

```
HTTP/1.1 302 Found
Date: Wed, 06 Dec 2023 00:15:29 GMT
Content-Length: 0
x-amz-cognito-request-id: 8aba6eb5-fb54-4bc6-9368-c3878434f0fb
Location: https://www.example.com?code=[Authorization code]
```

AWS Management Console

To configure an IdP for IdP-initiated SAML

1. Create a user pool, app client, and SAML identity provider.
2. Disassociate all social and OIDC identity providers from your app client, if any are associated.
3. Navigate to the **Social and external providers** menu of your user pool.
4. Edit or add a SAML provider.
5. Under **IdP-initiated SAML sign-in**, choose **Accept SP-initiated and and IdP-initiated SAML assertions**.
6. Choose **Save changes**.

API/CLI

To configure an IdP for IdP-initiated SAML

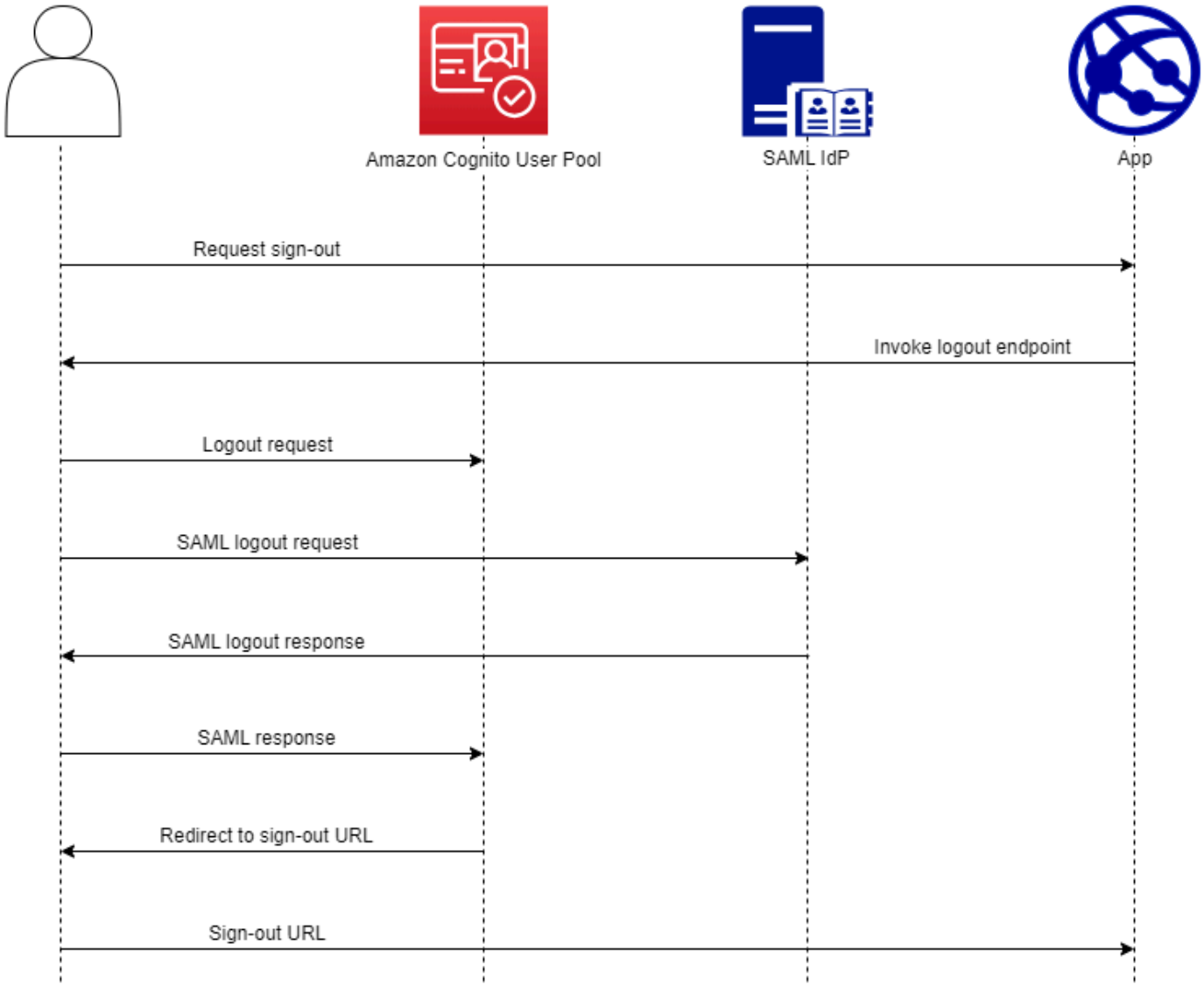
Configure IdP-initiated SAML with the `IDPInit` parameter in a CreateIdentityProvider or UpdateIdentityProvider API request. The following is an example `ProviderDetails` of an IdP that supports IdP-initiated SAML.

```
"ProviderDetails": {
      "MetadataURL" : "https://myidp.example.com/saml/metadata",
      "IDPSignout" : "true",
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

Signing out SAML users with single sign-out

Amazon Cognito supports SAML 2.0 single logout (SLO. With SLO, your application can sign out users from their SAML identity providers (IdPs) when they sign out from your user pool. This way, when users want to sign in to your application again, they must authenticate with their SAML IdP. Otherwise, they might have IdP or user pool browser cookies in place that pass them through to your application without the requirement that they provide credentials.

When you configure your SAML IdP to support **Sign-out flow**, Amazon Cognito redirects your user with a signed SAML logout request to your IdP. Amazon Cognito determines the redirect location from the `SingleLogoutService` URL in your IdP metadata. Amazon Cognito signs the sign-out request with your user pool signing certificate.



When you direct a user with a SAML session to your user pool `/logout` endpoint, Amazon Cognito redirects your SAML user with the following request to the SLO endpoint that's specified in the IdP metadata.

```
https://[SingleLogoutService endpoint]?
SAMLRequest=[encoded SAML request]&
RelayState=[RelayState]&
SigAlg=http://www.w3.org/2001/04/xmldsig-more#rsa-sha256&
Signature=[User pool RSA signature]
```

Your user then returns to your `saml2/logout` endpoint with a `LogoutResponse` from their IdP. Your IdP must send the `LogoutResponse` in an `HTTP POST` request. Amazon Cognito then redirects them to the redirect destination from their initial sign-out request.

Your SAML provider might send a `LogoutResponse` with more than one `AuthnStatement` in it. The `sessionIndex` in the first `AuthnStatement` in a response of this type must match the `sessionIndex` in the SAML response that originally authenticated the user. If the `sessionIndex` is in any other `AuthnStatement`, Amazon Cognito won't recognize the session and your user won't be signed out.

AWS Management Console

To configure SAML sign-out

1. Create a user pool, app client, and SAML IdP.
2. When you create or edit your SAML identity provider, under **Identity provider information**, check the box with the title **Add sign-out flow**.
3. From the **Social and external providers** menu of your user pool, choose your IdP and locate the **Signing certificate**.
4. Choose **Download as .crt**.
5. Configure your SAML provider to support SAML single logout and request signing, and upload the user pool signing certificate. Your IdP must redirect to `/saml2/logout` in your user pool domain.

API/CLI

To configure SAML sign-out

Configure single logout with the `IDPSignout` parameter of a CreateIdentityProvider or UpdateIdentityProvider API request. The following is an example `ProviderDetails` of an IdP that supports SAML single logout.

```
"ProviderDetails": {
      "MetadataURL" : "https://myidp.example.com/saml/metadata",
      "IDPSignout" : "true",,
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

SAML signing and encryption

SAML 2.0 sign-in is built around the user of an application as a bearer of requests and responses in their authentication flow. You might want to ensure that users aren't reading or modifying these SAML documents in transit. To accomplish this, add SAML signing and encryption to the SAML identity providers (IdPs) in your user pool. With SAML signing, your user pools adds a signature to SAML sign-in and sign-out requests. With your user pool public key, your IdP can verify that it's receiving unmodified SAML requests. Then, when your IdP responds and passes SAML assertions to users' browser sessions, the IdP can encrypt that response so that the user can't inspect their own attributes and entitlements.

With SAML signing and encryption, all cryptographic operations during user pool SAML operations must generate signatures and ciphertext with user-pool-provided keys that Amazon Cognito generates. Currently, you can't configure a user pool to sign requests or accept encrypted assertions with an external key.

> **Note**
>
> Your user pool certificates are valid for 10 years. Once per year, Amazon Cognito generates new signing and encryption certificates for your user pool. Amazon Cognito returns the most recent certificate when you request the signing certificate, and signs requests with the most recent signing certificate. Your IdP can encrypt SAML assertions with any user pool encryption certificate that isn't expired. Your previous certificates continue to be valid for their entire duration and the public key doesn't change between certificates. As a best practice, update the certificate in your provider configuration annually.

Topics

- Accepting encrypted SAML responses from your IdP
- Signing SAML requests

Accepting encrypted SAML responses from your IdP

Amazon Cognito and your IdP can establish confidentiality in SAML responses when users sign in and sign out. Amazon Cognito assigns a public-private RSA key pair and a certificate to each external SAML provider that you configure in your user pool. When you enable response encryption for your user pool SAML provider, you must upload your certificate to an IdP that supports encrypted SAML responses. Your user pool connection to your SAML IdP isn't functional before your IdP begins to encrypt all SAML assertions with the provided key.

The following is an overview of the flow of an encrypted SAML sign-in.

1. Your user starts sign-in and chooses their SAML IdP.
2. Your user pool Authorize endpoint redirects your user to their SAML IdP with a SAML sign-in request. Your user pool can optionally accompany this request with a signature that enables integrity verification by the IdP. When you want to sign SAML requests, you must configure your IdP to accept requests that your user pool has signed with the public key in the signing certificate.
3. The SAML IdP signs in your user and generates a SAML response. The IdP encrypts the response with the public key and redirects your user to your user pool `/saml2/idpresponse` endpoint. The IdP must encrypt the response as defined by the SAML 2.0 specification. For more information, see `Element <EncryptedAssertion>` in Assertions and Protocols for the OASIS Security Assertion Markup Language (SAML) V2.0.
4. Your user pool decrypts the ciphertext in the SAML response with the private key and signs in your user.

> **Important**
>
> When you enable response encryption for a SAML IdP in your user pool, your IdP must encrypt all responses with a public key that's specific to the provider. Amazon Cognito doesn't accept unencrypted SAML responses from a SAML external IdP that you configure to support encryption.

Any external SAML IdP in your user pool can support response encryption, and each IdP receives its own key pair.

AWS Management Console

To configure SAML response encryption

1. Create a user pool, app client, and SAML IdP.
2. When you create or edit your SAML identity provider, under **Sign requests and encrypt responses**, check the box with the title **Require encrypted SAML assertions from this provider**.
3. From the **Social and external providers** menu of your user pool, select your SAML IdP and choose **View encryption certificate**.
4. Choose **Download as .crt** and provide the downloaded file to your SAML IdP. Configure your SAML IdP to encrypt SAML responses with the key in the certificate.

API/CLI

To configure SAML response encryption

Configure response encryption with the `EncryptedResponses` parameter of a CreateIdentityProvider or UpdateIdentityProvider API request. The following is an example `ProviderDetails` of an IdP that supports request signing.

```
"ProviderDetails": {
      "MetadataURL" : "https://myidp.example.com/saml/metadata",
      "IDPSignout" : "true",
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

To get the encryption certificate from your user pool, make a DescribeIdentityProvider API request and retrieve the value of `ActiveEncryptionCertificate` in the response parameter `ProviderDetails`. Save this certificate and provide it to your IdP as the encryption certificate for sign-in requests from your user pool.

Signing SAML requests

The ability to prove the integrity of SAML 2.0 requests to your IdP is a security advantage of Amazon Cognito SP-initiated SAML sign-in. Each user pool with a domain receives a user pool X.509 signing certificate. With the public key in this certificate, user pools apply a cryptographic signature to the *sign-out requests* that your user pool generates when your users select a SAML IdP. You can optionally configure your app client to sign SAML *sign-in requests*. When you sign your SAML requests, your IdP can check that the signature in the XML metadata of your requests matches the public key in the user pool certificate that you provide.

AWS Management Console

To configure SAML request signing

1. Create a user pool, app client, and SAML IdP.
2. When you create or edit your SAML identity provider, under **Sign requests and encrypt responses**, check the box with the title **Sign SAML requests to this provider**.
3. From the **Social and external providers** menu of your user pool, choose **View signing certificate**.
4. Choose **Download as .crt** and provide the downloaded file to your SAML IdP. Configure your SAML IdP to verify the signature of incoming SAML requests.

API/CLI

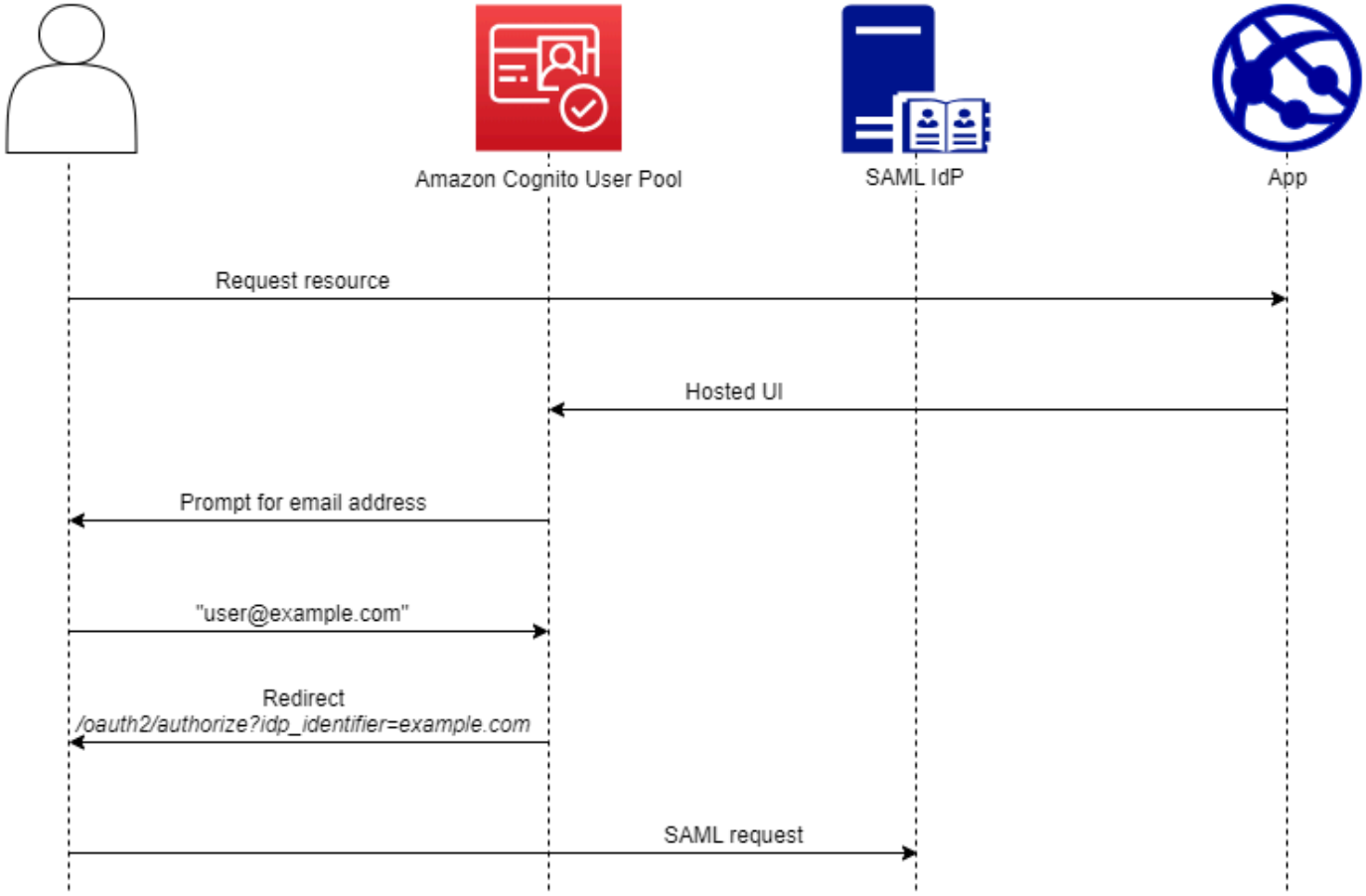
To configure SAML request signing

Configure request signing with the `RequestSigningAlgorithm` parameter of a CreateIdentityProvider or UpdateIdentityProvider API request. The following is an example `ProviderDetails` of an IdP that supports request signing.

```
"ProviderDetails": {
      "MetadataURL" : "https://myidp.example.com/saml/metadata",
      "IDPSignout" : "true",
      "RequestSigningAlgorithm" : "rsa-sha256",
      "EncryptedResponses" : "true",
      "IDPInit" : "true"
}
```

SAML identity provider names and identifiers

When you name your SAML identity providers (IdPs) and assign IdP identifiers, you can automate the flow of SP-initiated sign-in and sign-out requests to that provider. For information about string constraints to the provider name, see the `ProviderName` property of CreateIdentityProvider.



You can also choose up to 50 identifiers for your SAML providers. An identifier is a friendly name for an IdP in your user pool, and must be unique within the user pool. If your SAML identifiers match your users' email domains, managed login requests each user's email address, evaluates the domain in their email address, and redirects them to the IdP that corresponds to their domain. Because the same organization can own multiple domains, a single IdP can have multiple identifiers.

Whether you use or don't use email-domain identifiers, you can use identifiers in a multi-tenant app to redirect users to the correct IdP. When you want to bypass managed login entirely, you can customize the links that you present to users such that they redirect through the Authorize endpoint directly to their IdP. To sign in your users with an identifier and redirect to their IdP,

include the identifier in the format `idp_identifier=`*`myidp.example.com`* in the request parameters of their initial authorization request.

Another method to pass a user through to your IdP is to populate the parameter `identity_provider` with the name of your IdP in the following URL format.

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
identity_provider=MySAMLIdP&
client_id=1example23456789&
redirect_uri=https://www.example.com
```

After a user signs in with your SAML IdP, your IdP redirects them with a SAML response in the `HTTP POST` body to your `/saml2/idpresponse` endpoint. Amazon Cognito processes the SAML assertion and, if the claims in the response meet expectations, redirects to your app client callback URL. After your user has completed authentication in this way, they have interacted with webpages for only your IdP and your app.

With IdP identifiers in a domain format, managed login requests email addresses at sign-in and then, when the email domain matches an IdP identifier, redirects users to the sign-in page for their IdP. As an example, you build an app that requires sign-in by employees of two different companies. The first company, AnyCompany A, owns `exampleA.com` and `exampleA.co.uk`. The second company, AnyCompany B, owns `exampleB.com`. For this example, you have set up two IdPs, one for each company, as follows:

- For IdP A, you define identifiers `exampleA.com` and `exampleA.co.uk`.
- For IdP B, you define identifier `exampleB.com`.

In your app, invoke managed login for your app client to prompt each user to enter their email address. Amazon Cognito derives the domain from the email address, correlates the domain to an IdP with a domain identifier, and redirects your user to the correct IdP with a request to the Authorize endpoint that contains an `idp_identifier` request parameter. For example, if a user enters bob@`exampleA.co.uk`, the next page that they interact with is the IdP sign-in page at `https://auth.exampleA.co.uk/sso/saml`.

You can also implement the same logic independently. In your app, you can build a custom form that collects user input and correlates it to the correct IdP according to your own logic. You can

generate custom portals for each of your app tenants, where each links to the authorize endpoint with the tenant's identifier in the request parameters.

To collect an email address and parse the domain in managed login, assign at least one identifier to each SAML IdP that you have assigned to your app client. By default, the managed login sign-in screen displays a button for each of the IdPs that you have assigned to your app client. However, if you have successfully assigned identifiers, your classic hosted UI sign-in page looks like the following image.
An Amazon Cognito managed login sign-in page displaying local user sign-in and a prompt for a federated user to enter an email address.

> **Note**
>
> In the classic hosted UI, the sign-in page for your app client automatically prompts for an email address when you assign identifiers to your IdPs. In the managed login experience, you must enable this behavior in the branding editor. In the **Authentication behavior** settings category, select **Domain search input** under the heading **Provider display**.

Domain parsing in managed login requires that you use domains as your IdP identifiers. If you assign an identifier of any type to each of the SAML IdPs for an app client, managed login for that app no longer displays IdP-selection buttons. Add IdP identifiers for SAML when you intend to use email parsing or custom logic to generate redirects. When you want to generate silent redirects and also want your managed login pages to display a list of IdPs, don't assign identifiers and use the `identity_provider` request parameter in your authorization requests.

- If you assign only one SAML IdP to your app client, the managed login sign-in page displays a button to sign in with that IdP.
- If you assign an identifier to every SAML IdP that you activate for your app client, a user input prompt for an email address appears in the managed login sign-in page.
- If you have multiple IdPs and you do not assign an identifier to all of them, the managed login sign-in page displays a button to sign in with each assigned IdP.
- If you assigned identifiers to your IdPs and you want your managed login pages to display a selection of IdP buttons, add a new IdP that has no identifier to your app client, or create a new app client. You can also delete an existing IdP and add it again without an identifier. If you create a new IdP, your SAML users will create new user profiles. This duplication of active users might have a billing impact in the month that you change your IdP configuration.

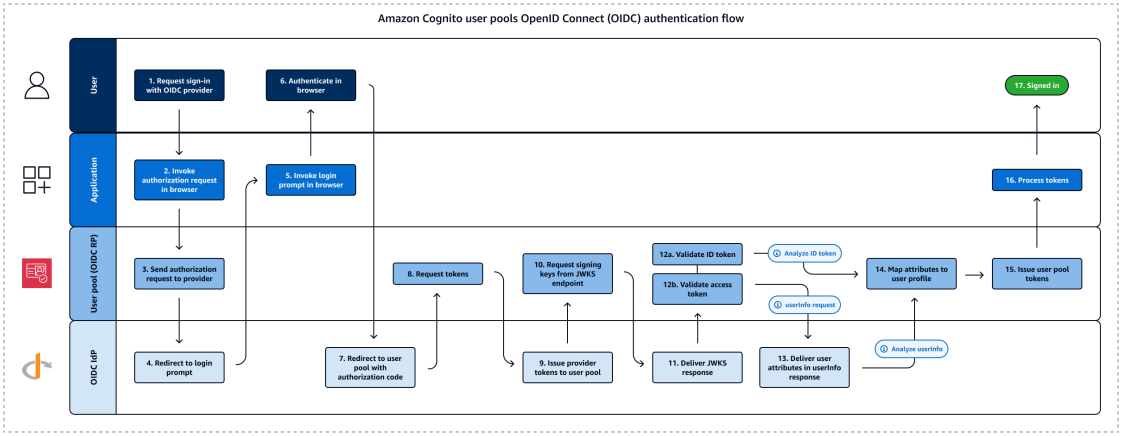
For more information about IdP setup, see Configuring identity providers for your user pool.

Using OIDC identity providers with a user pool

Users can sign in to your application using their existing accounts from OpenID Connect (OIDC) identity providers (IdPs). With OIDC providers, users of independent single sign-on systems can provide existing credentials while your application receives OIDC tokens in the shared format of user pools. To configure an OIDC IdP, set up your IdP to handle your user pool as the RP and configure your application to handle your user pool as the IdP. Amazon Cognito serves as an intermediate step between multiple OIDC IdPs and your applications. Your user pool applies attribute-mapping rules to the claims in the ID and access tokens that your provider passes directly to your user pool. Amazon Cognito then issues new tokens based on the mapped user attributes and any additional adjustments you've made to the authentication flow with Lambda triggers.

Users who sign in with an OIDC IdP aren't required to provide new credentials or information to access your user pool application. Your application can silently redirect them to their IdP for sign-in, with a user pool as a tool in the background that standardizes the token format for your application. To learn more about IdP redirection, see Authorize endpoint.

Like with other third-party identity providers, you must register your application with the OIDC provider and obtain information about the IdP application that you want to connect to your user pool. A user pool OIDC IdP requires a client ID, client secret, scopes that you want to request, and information about provider service endpoints. Your user pool can discover the provider OIDC endpoints from a discovery endpoint or you can enter them manually. You must also examine provider ID tokens and create attribute mappings between the IdP and the attributes in your user pool.



See OIDC user pool IdP authentication flow for more details about this authentication flow.

> **Note**
>
> Sign-in through a third party (federation) is available in Amazon Cognito user pools. This feature is independent of OIDC federation with Amazon Cognito identity pools.

You can add an OIDC IdP to your user pool in the AWS Management Console, through the AWS CLI, or with the user pool API method CreateIdentityProvider.

Topics

- Prerequisites
- Register an application with an OIDC IdP
- Add an OIDC IdP to your user pool
- Test your OIDC IdP configuration
- OIDC user pool IdP authentication flow
- Step-up authentication for federated users

Prerequisites

Before you begin, you need the following:

- A user pool with an app client and a user pool domain. For more information, see Create a user pool.
- An OIDC IdP with the following configuration:
  - Supports `client_secret_post` client authentication. Amazon Cognito doesn't check the `token_endpoint_auth_methods_supported` claim at the OIDC discovery endpoint for your IdP. Amazon Cognito doesn't support `client_secret_basic` client authentication. For more information on client authentication, see Client Authentication in the OpenID Connect documentation.
  - Only uses HTTPS for OIDC endpoints such as `openid_configuration`, `userInfo`, and `jwks_uri`.
  - Only uses TCP ports 80 and 443 for OIDC endpoints.
  - Only signs ID tokens with HMAC-SHA, ECDSA, or RSA algorithms.
  - Publishes a key ID `kid` claim at its `jwks_uri` and includes a `kid` claim in its tokens.

- Presents a non-expired public key with a valid root CA trust chain.

Register an application with an OIDC IdP

Before you add an OIDC IdP to your user pool configuration and assign it to app clients, you set up an OIDC client application in your IdP. Your user pool is the relying-party application that will manage authentication with your IdP.

To register with an OIDC IdP

1. Create a developer account with the OIDC IdP.

   Links to OIDC IdPs

OIDC IdP	How to Install	OIDC Discovery URL
Salesforce	Salesforce as an OpenID Connect Identity Provider	`https://MyDomainName.my.salesforce.com/.well-known/openid-configuration`
OneLogin	Connect an OIDC enabled app	`https://your-domain.onelogin.com/oidc/2/.well-known/openid-configuration`
JumpCloud	SSO with OIDC	`https://oauth.id.jumpcloud.com/.well-known/openid-configuration`
Okta	Install an Okta identity provider	`https://Your Okta subdomain.okta.com/.well-known/openid-configuration`
Microsoft Entra ID	OpenID Connect on the Microsoft identity platform	`https://login.microsoftonline.com/{tenant}/v2.0`

OIDC IdP	How to Install	OIDC Discovery URL
		Values of `tenant` can include a tenant ID, `common`, `organizations` , or `consumers` .

2. Register your user pool domain URL with the `/oauth2/idpresponse` endpoint with your OIDC IdP. This ensures that the OIDC IdP later accepts it from Amazon Cognito when it authenticates users.

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
```

3. Select the scopes that you want your user directory to share with your user pool. The scope **openid** is required for OIDC IdPs to offer any user information. The `email` scope is needed to grant access to the `email` and `email_verified` claims. Additional scopes in the OIDC specification are `profile` for all user attributes and phone for `phone_number` and `phone_number_verified`.
4. The OIDC IdP provides you with a client ID and a client secret. Note these values and add them to the configuration of the OIDC IdP that you add to your user pool later.

Example: Use Salesforce as an OIDC IdP with your user pool

You use an OIDC IdP when you want to establish trust between an OIDC-compatible IdP such as Salesforce and your user pool.

1. Create an account on the Salesforce Developers website.
2. Sign in through your developer account that you set up in the previous step.
3. From your Salesforce page, do one of the following:
   - If you're using Lightning Experience, choose the setup gear icon, then choose **Setup Home**.
   - If you're using Salesforce Classic and you see **Setup** in the user interface header, choose it.
   - If you're using Salesforce Classic and you don't see **Setup** in the header, choose your name from the top navigation bar, and choose **Setup** from the drop-down list.
4. On the left navigation bar, choose **Company Settings**.
5. On the navigation bar, choose **Domain**, enter a domain, and choose **Create**.
6. On the left navigation bar, under **Platform Tools**, choose **Apps**.

7. Choose **App Manager**.
8. a. Choose **New connected app**.

   b. Complete the required fields.

   Under **Start URL**, enter a URL at the `/authorize` endpoint for the user pool domain that signs in with your Salesforce IdP. When your users access your connected app, Salesforce directs them to this URL to complete sign-in. Then Salesforce redirects the users to the callback URL that you have associated with your app client.

   ```
   https://mydomain.auth.us-east-1.amazoncognito.com/authorize?
   response_type=code&client_id=<your_client_id>&redirect_uri=https://
   www.example.com&identity_provider=CorpSalesforce
   ```

   c. Enable **OAuth settings** and enter the URL of the `/oauth2/idpresponse` endpoint for your user pool domain in **Callback URL**. This is the URL where Salesforce issues the authorization code that Amazon Cognito exchanges for an OAuth token.

   ```
   https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/idpresponse
   ```

9. Select your scopes. You must include the scope **openid**. To grant access to the **email** and **email_verified** claims, add the **email** scope. Separate scopes by spaces.
10. Choose **Create**.

    In Salesforce, the client ID is called a **Consumer Key**, and the client secret is a **Consumer Secret**. Note your client ID and client secret. You will use them in the next section.

Add an OIDC IdP to your user pool

After you set up your IdP, you can configure your user pool to handle authentication requests with an OIDC IdP.

Amazon Cognito console

Add an OIDC IdP in the console

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools** from the navigation menu.
3. Choose an existing user pool from the list, or create a user pool.

4. Choose the **Social and external providers** menu and then select **Add an identity provider**.
5. Choose an **OpenID Connect** IdP.
6. Enter a unique **Provider name**.
7. Enter the IdP **Client ID**. This is the ID of the application client you build in your OIDC IdP. The client ID that you provide must be an OIDC provider that you've configured with a callback url of `https://[your user pool domain]/oauth2/idpresponse`.
8. Enter the IdP **Client secret**. This must be the client secret for the same application client from the previous step.
9. Enter **Authorized scopes** for this provider. Scopes define which groups of user attributes (such as `name` and `email`) that your application will request from your provider. Scopes must be separated by spaces, following the OAuth 2.0 specification.

   Your IdP might prompt users to consent to providing these attributes to your application when they sign in.
10. Choose an **Attribute request method**. IdPs might require that requests to their `userInfo` endpoints are formatted as either `GET` or `POST`. The Amazon Cognito `userInfo` endpoint requires `HTTP GET` requests, for example.
11. Choose a **Setup method** for the way that you want your user pool to determine the path to key OIDC-federation endpoints at your IdP. Typically, IdPs host a `/well-known/openid-configuration` endpoint at an issuer base URL. If this is the case for your provider, the **Auto fill through issuer URL** option prompts you for that base URL, attempts to access the `/well-known/openid-configuration` path from there, and reads the endpoints listed there. You might have non-typical endpoint paths or wish to pass requests to one or more endpoints through an alternate proxy. In this case, select **Manual input** and specify paths for the `authorization`, `token`, `userInfo`, and `jwks_uri` endpoints.

> **Note**
>
> The URL should start with `https://`, and shouldn't end with a slash `/`. Only port numbers 443 and 80 can be used with this URL. For example, Salesforce uses this URL:
> `https://login.salesforce.com`
> If you choose auto fill, the discovery document must use HTTPS for the following values: `authorization_endpoint`, `token_endpoint`, `userinfo_endpoint`, and `jwks_uri`. Otherwise the login will fail.

12. Configure your attribute-mapping rules under **Map attributes between your OpenID Connect provider and your user pool**. **User pool attribute** is the *destination* attribute in the Amazon Cognito user profile and **OpenID Connect attribute** is the *source* attribute that you want Amazon Cognito to find in an ID-token claim or `userInfo` response. Amazon Cognito automatically maps the OIDC claim **sub** to `username` in the destination user profile.

    For more information, see Mapping IdP attributes to profiles and tokens.
13. Choose **Add identity provider**.
14. From the **App clients** menu, select an app client from the list. Navigate to the **Login pages** tab and under **Managed login pages configuration**, select **Edit**. Locate **Identity providers** and add your new OIDC IdP.
15. Choose **Save changes**.

API/CLI

See the OIDC configuration in example two at CreateIdentityProvider. You can modify this syntax and use it as the request body of `CreateIdentityProvider`, `UpdateIdentityProvider`, or the `--cli-input-json` input file for create-identity-provider.

Test your OIDC IdP configuration

In your application, you must invoke a browser in the user's client so that they can sign in with their OIDC provider. Test sign-in with your provider after you have completed the setup procedures in the preceding sections. The following example URL loads the sign-in page for your user pool with a prefix domain.

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://www.example.com
```

This link is the page that Amazon Cognito directs you to when you go to the **App clients** menu, select an app client, navigate to the **Login pages** tab, and select **View login page**. For more information about user pool domains, see Configuring a user pool domain. For more information about app clients, including client IDs and callback URLs, see Application-specific settings with app clients.

The following example link sets up silent redirect to the `MyOIDCIdP` provider from the Authorize endpoint with an `identity_provider` query parameter. This URL bypasses interactive user pool sign-in with managed login and goes directly to the IdP sign-in page.

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
identity_provider=MyOIDCIdP&response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com
```

OIDC user pool IdP authentication flow

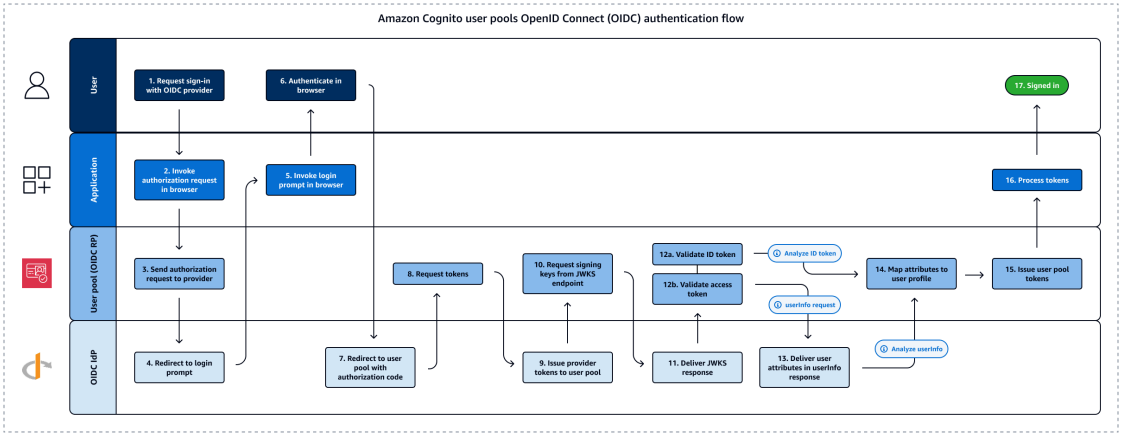
With OpenID Connect (OIDC) sign-in, your user pool automates an authorization-code sign-in flow with your identity provider (IdP). After your user completes sign-in with their IdP, Amazon Cognito collects their code at the `oauth2/idpresponse` endpoint of the external provider. With the resulting access token, your user pool queries the IdP `userInfo` endpoint to retrieve user attributes. Your user pool then compares the received attributes to the attribute-mapping rules you've set up and populates the user's profile and ID token accordingly.

The OAuth 2.0 scopes that you request in your OIDC provider configuration define the user attributes that the IdP provides to Amazon Cognito. As a best security practice, only request the scopes that correspond to attributes that you want to map to your user pool. For example, if your user pool requests `openid profile`, you'll get all possible attributes, but if you request `openid email phone_number` you'll only get the user's email address and phone number. You can configure the scopes that you request from OIDC IdPs to differ from those you authorize and request in the app client and user pool authentication request.

When your user signs in to your application using an OIDC IdP, your user pool conducts the following authentication flow.

1. A user accesses your managed login sign-in page, and chooses to sign in with their OIDC IdP.
2. Your application directs the user's browser to the authorization endpoint of your user pool.
3. Your user pool redirects the request to the authorization endpoint of the OIDC IdP.
4. Your IdP displays a login prompt.
5. In your application, your user's session displays a sign-in prompt for the OIDC IdP.
6. The user enters their credentials for the IdP or presents a cookie for an already-authenticated session.
7. After your user authenticates, the OIDC IdP redirects to Amazon Cognito with an authorization code.

8. Your user pool exchanges the authorization code for ID and access tokens. Amazon Cognito receives access tokens when you configure your IdP with the scopes `openid`. The claims in the ID token and the `userInfo` response are determined by additional scopes from your IdP configuration, for example `profile` and `email`.
9. Your IdP issues the requested tokens.
10. Your user pool determines the path to the IdP `jwks_uri` endpoint from the issuer URLs in your IdP configuration and requests the token signing keys from the JSON web key set (JWKS) endpoint.
11. The IdP returns signing keys from the JWKS endpoint.
12. Your user pool validates the IdP tokens from signature and expiration data in the tokens.
13. Your user pool authorizes a request to the IdP `userInfo` endpoint with the access token. The IdP responds with user data based on the access token scopes.
14. Your user pool compares the ID token and `userInfo` response from the IdP to the attribute-mapping rules in your user pool. It writes mapped IdP attributes to user pool profile attributes.
15. Amazon Cognito issues your application bearer tokens, which might include identity, access, and refresh tokens.
16. Your application processes the user pool tokens and signs the user in.



> **Note**
>
> Amazon Cognito cancels authentication requests that do not complete within 5 minutes, and redirects the user to managed login. The page displays a `Something went wrong` error message.

OIDC is an identity layer on top of OAuth 2.0, which specifies JSON-formatted (JWT) identity tokens that are issued by IdPs to OIDC client apps (relying parties). See the documentation for your OIDC IdP for information about to add Amazon Cognito as an OIDC relying party.

When a user authenticates with an authorization code grant, the user pool returns ID, access, and refresh tokens. The ID token is a standard OIDC token for identity management, and the access token is a standard OAuth 2.0 token. For more information about grant types that your user pool app client can support, see Authorize endpoint.

How a user pool processes claims from an OIDC provider

When your user completes sign-in with a third-party OIDC provider, managed login retrieves an authorization code from the IdP. Your user pool exchanges the authorization code for access and ID tokens with the `token` endpoint of your IdP. Your user pool doesn't pass these tokens on to your user or your app, but uses them to build a user profile with data that it presents in claims in its own tokens.

Amazon Cognito doesn't independently validate the access token. Instead, it requests user-attribute information from the provider `userInfo` endpoint and expects the request to be denied if the token isn't valid.

Amazon Cognito validates the provider ID token with the following checks:

1. Check that the provider signed the token with an algorithm from the following set: RSA, HMAC, Elliptic Curve.
2. If the provider signed the token with an asymmetric signing algorithm, check that the signing key ID in the token `kid` claim is listed at the provider `jwks_uri` endpoint. Amazon Cognito refreshes the signing key from the JWKS endpoint in your IdP configuration for each IdP ID token that it processes.
3. Compare the ID token signature to the signature that it expects based on provider metadata.
4. Compare the `iss` claim to the OIDC issuer configured for the IdP.
5. Compare the `aud` claim matches the client ID configured on the IdP, or that it contains the configured client ID if there are multiple values in the `aud` claim.
6. Check that the timestamp in the `exp` claim is not before the current time.

Your user pool validates the ID token, then attempts a request to the provider `userInfo` endpoint with the provider access token. It retrieves any user profile information that the scopes in the access token authorize it to read. Your user pool then searches for the user attributes that you

have set as required in your user pool. You must create attribute mappings in your provider configuration for required attributes. Your user pool checks the provider ID token and the `userInfo` response. Your user pool writes all claims that match mapping rules to user attributes on the user pool user profile. Your user pool ignores attributes that match a mapping rule but aren't required and aren't found in the provider's claims.

Step-up authentication for federated users

For federated users who sign in through an OpenID Connect (OIDC) identity provider (IdP), Amazon Cognito passes through the ACR and AMR values of the OIDC IdP instead of computing its own. The OIDC IdP owns authentication for these federated users. Step-up authentication for a federated user happens through the OIDC IdP: Amazon Cognito redirects to the IdP with `acr_values`. For more information about ACR levels and AMR values, see the section called “Authentication levels with ACR and AMR claims”.

You map the ACR values of the OIDC IdP to Amazon Cognito levels with the `AcrMapping` field on the IdP. Configure this field with the CreateIdentityProvider or UpdateIdentityProvider operation. Amazon Cognito maps the ACR value of the OIDC IdP to a Amazon Cognito level, then stamps the corresponding level name that you configured for your user pool onto the Amazon Cognito token. The following example configures an `AcrMapping` for an OIDC provider.

```
{
  "ProviderName": "MyOktaIdP",
  "ProviderType": "OIDC",
  "AcrMapping": {
    "Level1": "urn:okta:loa:1",
    "Level2": "urn:okta:loa:2",
    "Level3": "urn:okta:loa:3",
    "Level4": "urn:okta:loa:4"
  }
}
```

Setting `AcrMapping` on the identity provider is available in all feature plans. It isn't restricted to the Essentials or Plus feature plan.

The following behavior applies to federated users who sign in through an OIDC IdP.

- If the OIDC IdP returns an ACR value that isn't in the configured `AcrMapping`, Amazon Cognito sets the token's `acr` claim to the lowest level (the default `urn:cognito:loa:1`, or the level-1 name that you configured for your user pool). This isn't an error.

- Amazon Cognito drops any AMR value that it doesn't recognize and retains only the known values.
- If the OIDC IdP doesn't support ACR and AMR, or returns no AMR values that Amazon Cognito recognizes, Amazon Cognito sets `fed` as a default placeholder AMR value.
- Amazon Cognito trusts the `acr` and `amr` values that the OIDC IdP asserts, subject to mapping and filtering. It doesn't validate which challenges the IdP actually performed.

In a federation flow, your application redirects to the `/oauth2/authorize` endpoint with the target level or levels expressed in the names of your user pool. Amazon Cognito translates the values to the values that the OIDC IdP expects with the `AcrMapping`, redirects to the IdP, receives the ACR and AMR values of the IdP, maps them back, and issues Amazon Cognito tokens. For more information about configuring an IdP, see Configuring identity providers.

Mapping IdP attributes to profiles and tokens

Identity provider (IdP) services, including Amazon Cognito, can typically record more information about a user. You might want to know what company they work for, how to contact them, and other identifying information. But the format that these attributes take has variations between providers. For example, set up three IdPs from three different vendors with your user pool and examine an example SAML assertion, ID token, or `userInfo` payload from each. One will represent the user's email address as `email`, another as `emailaddress`, and the third as `http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`.

A major benefit that comes from consolidation of IdPs with a user pool is the ability to map the variety of attribute names into a single OIDC token schema with consistent, predicable, shared attribute names. This way, your developers aren't required to maintain the logic for processing a complex variety of single sign-on events. This format consolidation is attribute mapping. User pool attribute mapping assigns IdP attribute names to the corresponding user pool attribute names. For example, you can configure your user pool to write the value of an `emailaddress` claim to the standard user pool attribute `email`.

Each user pool IdP has a separate attribute-mapping schema. To specify attribute mappings for your IdP, configure a user pool identity provider in the Amazon Cognito console, an AWS SDK, or the user pools REST API .

Things to know about mappings

Before you begin to set up user-attribute mapping, review the following important details.

- Before you can map an IdP claim to a custom attribute, that custom attribute must already exist in your user pool schema. You define custom attributes when you create the user pool, or add them afterward, and you can't map an IdP attribute to a custom attribute that you haven't yet added to the schema. For more information about defining attributes in your pool, see Working with user attributes.
- When a federated user signs in to your application, a mapping must be present for each user pool attribute that your user pool requires. For example, if your user pool requires an `email` attribute for sign-up, map this attribute to its equivalent from the IdP.
- By default, mapped email addresses are unverified. You can't verify a mapped email address using a one-time code. Instead, map an attribute from your IdP to get the verification status. For example, Google and most OIDC providers include the `email_verified` attribute.
- You can map identity provider (IdP) tokens to custom attributes in your user pool. Social providers present an access token, and OIDC providers present an access and ID token. To map a token, add a custom attribute with a maximum length of 2,048 characters, grant your app client write access to the attribute, and map `access_token` or `id_token` from the IdP to the custom attribute.
- For each mapped user pool attribute, the maximum value length of 2,048 characters must be large enough for the value that Amazon Cognito obtains from the IdP. Otherwise, Amazon Cognito reports an error when users sign in to your application. Amazon Cognito doesn't support mapping IdP tokens to custom attributes when the tokens are more than 2,048 characters long.
- Amazon Cognito derives the `username` attribute in a federated user's profile from specific claims that your federated IdP passes, as shown in the following table. Amazon Cognito prepends this attribute value with the name of your IdP, for example `MyOIDCIdP_[sub]`. When you want your federated users to have an attribute that exactly matches an attribute in your external user directory, map that attribute to a Amazon Cognito sign-in attribute like `preferred_username`.

Identity Provider	`username` source attribute
Facebook	`id`
Google	`sub`
Login with Amazon	`user_id`
Sign in with Apple	`sub`

Identity Provider	`username` source attribute
SAML providers	`NameID`
OpenID Connect (OIDC) providers	`sub`

- When a user pool is case insensitive, Amazon Cognito converts the entire automatically generated username of a federated user to lowercase, including the IdP name prefix. The following is an example username for a case-sensitive user pool: `MySAML_TestUser@example.com`. The following is the same username for a case-*insensitive* user pool: `mysaml_testuser@example.com`.

  In case-insensitive user pools, your Lambda triggers that process the username must account for changes to the entire username. These changes include any mixed-case values from the username source attributes that you mapped from your IdP. To link your IdP to a user pool with a different case-sensitivity setting than your current pool, create a new user pool.
- Amazon Cognito must be able to update your mapped user pool attributes when users sign in to your application. When a user signs in through an IdP, Amazon Cognito updates the mapped attributes with the latest information from the IdP. Amazon Cognito only updates mapped attributes when their values change. To ensure that Amazon Cognito can update the attributes, check the following requirements:
  - All of the user pool custom attributes that you map from your IdP must be *mutable*. You can update mutable custom attributes at any time. By contrast, you can only set a value for a user's *immutable* custom attribute when you first create the user profile. To create a mutable custom attribute in the Amazon Cognito console, activate the **Mutable** checkbox for the attribute you add when you select **Add custom attributes** in the **Sign-up** menu. Or, if you create your user pool by using the CreateUserPool API operation, you can set the `Mutable` parameter for each of these attributes to `true`. If your IdP sends a value for a mapped immutable attribute, Amazon Cognito returns an error and sign-in fails.
  - In the app client settings for your application, the mapped attributes must be *writable*. You can set which attributes are writable in the **App clients** page in the Amazon Cognito console. Or, if you create the app client by using the `CreateUserPoolClient` API operation, you can add these attributes to the `WriteAttributes` array. If your IdP sends a value for a mapped non-writable attribute, Amazon Cognito doesn't set the attribute value and proceeds with authentication.
- When IdP attributes contain multiple values, Amazon Cognito flattens all values into a single comma-delimited string enclosed in the square-bracket characters `[` and `]`. Amazon Cognito URL

form-encodes the values containing non-alphanumeric characters except for `.`, `-`, `*`, and `_`. You must decode and parse the individual values before you use them in your app.

- The destination attribute retains any value that your attribute-mapping rules assign to it unless a sign-in or administrative action changes it. Amazon Cognito doesn't remove attributes from users when the source attribute is no longer sent in the provider token or SAML assertion. The following actions remove the value of an attribute from a user pool profile for a federated user:
  1. The IdP sends a blank value for the source attribute and a mapping rule applies the blank value to the destination attribute.
  2. You clear the value of the mapped attribute with an DeleteUserAttributes or AdminDeleteUserAttributes request.

Specifying identity provider attribute mappings for your user pool (AWS Management Console)

You can use the AWS Management Console to specify attribute mappings for the IdP your user pool.

> **Note**
>
> Amazon Cognito will map incoming claims to user pool attributes only if the claims exist in the incoming token. If a previously mapped claim no longer exists in the incoming token, it won't be deleted or changed. If your application requires mapping of deleted claims, you can use the Pre-Authentication Lambda trigger to delete the custom attribute during authentication and allow these attributes to repopulate from the incoming token.

To specify a social IdP attribute mapping

1. Sign in to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Choose **Add an identity provider**, or choose the **Facebook**, **Google**, **Amazon** or **Apple** IdP you have configured. Locate **Attribute mapping** and choose **Edit**.

   For more information about adding a social IdP, see Using social identity providers with a user pool.
5. For each attribute you need to map, complete the following steps:

a. Select an attribute from the **User pool attribute** column. This is the attribute that is assigned to the user profile in your user pool. Custom attributes are listed after standard attributes.

b. Select an attribute from the ***<provider>*** **attribute** column. This will be the attribute passed from the provider directory. Known attributes from the social provider are provided in a drop-down list.

c. To map additional attributes between your IdP and Amazon Cognito, choose **Add another attribute**.

6. Choose **Save changes**.

To specify a SAML provider attribute mapping

1. Sign in to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Social and external providers** menu.
4. Choose **Add an identity provider**, or choose the SAML IdP you have configured. Locate **Attribute mapping**, and choose **Edit**. For more information about adding a SAML IdP, see Using SAML identity providers with a user pool.
5. For each attribute you need to map, complete the following steps:

   a. Select an attribute from the **User pool attribute** column. This is the attribute that is assigned to the user profile in your user pool. Custom attributes are listed after standard attributes.

   b. Select an attribute from the **SAML attribute** column. This will be the attribute passed from the provider directory.

      Your IdP might offer sample SAML assertions for reference. Some IdPs use simple names, such as `email`, while others use URL-formatted attribute names similar to:

      ```
      http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress
      ```

   c. To map additional attributes between your IdP and Amazon Cognito, choose **Add another attribute**.

6. Choose **Save changes**.

Specifying identity provider attribute mappings for your user pool (AWS CLI and AWS API)

The following request body for CreateIdentityProvider or UpdateIdentityProvider maps the SAML provider "MyIdP" attributes `emailaddress`, `birthdate`, and phone to the user pool attributes `email`, `birthdate`, and phone_number, in that order. This is a complete request body for a SAML 2.0 provider—your request body will vary depending on IdP type and specific details. The attribute mapping is in the `AttributeMapping` parameter.

```
{
   "AttributeMapping": {
      "email" : "emailaddress",
      "birthdate" : "birthdate",
      "phone_number" : "phone"
   },
   "IdpIdentifiers": [
      "IdP1",
      "pdxsaml"
   ],
   "ProviderDetails": {
      "IDPInit": "true",
      "IDPSignout": "true",
      "EncryptedResponses" : "true",
      "MetadataURL": "https://auth.example.com/sso/saml/metadata",
      "RequestSigningAlgorithm": "rsa-sha256"
   },
   "ProviderName": "MyIdP",
   "ProviderType": "SAML",
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Use the following commands to specify IdP attribute mappings for your user pool.

To specify attribute mappings at provider creation time

- AWS CLI: `aws cognito-idp create-identity-provider`

  Example with metadata file: `aws cognito-idp create-identity-provider --user-pool-id <user_pool_id> --provider-name=SAML_provider_1 --provider-type SAML --provider-details file:///details.json --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`

  Where `details.json` contains:

```
{
    "MetadataFile": "<SAML metadata XML>"
}
```

Note

If the *<SAML metadata XML>* contains any quotations ("), they must be escaped (\").

Example with metadata URL:

```
aws cognito-idp create-identity-provider \
--user-pool-id us-east-1_EXAMPLE \
--provider-name=SAML_provider_1 \
--provider-type SAML \
--provider-details MetadataURL=https://myidp.example.com/saml/metadata \
--attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/
emailaddress
```

- API/SDK: CreateIdentityProvider

To specify attribute mappings for an existing IdP

- AWS CLI: `aws cognito-idp update-identity-provider`

  Example: `aws cognito-idp update-identity-provider --user-pool-id <user_pool_id> --provider-name <provider_name> --attribute-mapping email=http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress`
- API/SDK: UpdateIdentityProvider

To get information about attribute mapping for a specific IdP

- AWS CLI: `aws cognito-idp describe-identity-provider`

  Example: `aws cognito-idp describe-identity-provider --user-pool-id <user_pool_id> --provider-name <provider_name>`
- API/SDK: DescribeIdentityProvider

Linking federated users to an existing user profile

Often, the same user has a profile with multiple identity providers (IdPs) that you have connected to your user pool. Amazon Cognito can link each occurrence of a user to the same user profile in your directory. This way, one person who has multiple IdP users can have a consistent experience in your app. AdminLinkProviderForUser tells Amazon Cognito to recognize a user's unique ID in your federated directory as a user in the user pool. A user in your user pool counts as one monthly active user (MAU) for the purposes of billing when you have zero or more federated identities associated with the user profile.

When a federated user signs in to your user pool for the first time, Amazon Cognito looks for a local profile that you have linked to their identity. If no linked profile exists, your user pool creates a new profile. You can create a local profile and link it to your federated user at any time before their first sign-in, in an `AdminLinkProviderForUser` API request, either in a planned prestaging task or in a Pre sign-up Lambda trigger. After your user signs in and Amazon Cognito detects a linked local profile, your user pool reads your user's claims and compares them to mapping rules for the IdP. Your user pool then updates the linked local profile with the claims mapped from their sign-in. In this way, you can configure the local profile with access claims and keep their identity claims up-to-date with your provider. After Amazon Cognito matches your federated user to a linked profile, they always sign in to that profile. You can then link more of your user's provider identities to the same profile, giving one customer a consistent experience in your app. To link a federated user who has previously signed in, you must first delete their existing profile. You can identify existing profiles by their format: *`[Provider name]`*`_identifier`. For example, `LoginWithAmazon_amzn1.account.AFAEXAMPLE`. A user that you created and then linked to a third-party user identity has the username that they were created with, and an `identities` attribute that contains the details of their linked identities.

> **Important**
>
> Because `AdminLinkProviderForUser` allows a user with an external federated identity to sign in as an existing user in the user pool, it is critical that it only be used with external IdPs and provider attributes that have been trusted by the application owner.

For example, if you're a managed service provider (MSP) with an app that you share with multiple customers. Each of the customers signs in to your app through Active Directory Federation Services (ADFS). Your IT administrator, Carlos, has an account in each of your customers' domains. You want Carlos to be recognized as an app administrator every time he signs in, regardless of the IdP.

Your ADFS IdPs present Carlos' email address `msp_carlos@example.com` in the `email` claim of the Carlos' SAML assertions to Amazon Cognito. You create a user in your user pool with the user name `Carlos`. The following AWS Command Line Interface (AWS CLI) commands link Carlos' identities from IdPs ADFS1, ADFS2, and ADFS3.

> **Note**
>
> You can link a user based on specific attribute claims. This ability is unique to OIDC and SAML IdPs. For other provider types, you must link based on a fixed source attribute. For more information, see AdminLinkProviderForUser. You must set `ProviderAttributeName` to `Cognito_Subject` when you link a social IdP to a user profile. `ProviderAttributeValue` must be the user's unique identifier with your IdP.

```
aws cognito-idp admin-link-provider-for-user \
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
 ProviderName=ADFS1,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com

aws cognito-idp admin-link-provider-for-user \
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
 ProviderName=ADFS2,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com

aws cognito-idp admin-link-provider-for-user \
--user-pool-id us-east-1_EXAMPLE \
--destination-user ProviderAttributeValue=Carlos,ProviderName=Cognito \
--source-user
 ProviderName=ADFS3,ProviderAttributeName=email,ProviderAttributeValue=msp_carlos@example.com
```

The user profile `Carlos` in your user pool now has the following `identities` attribute.

```
[{
    "userId": "msp_carlos@example.com",
    "providerName": "ADFS1",
    "providerType": "SAML",
    "issuer": "http://auth.example.com",
    "primary": false,
    "dateCreated": 111111111111111
```

```
}, {
    "userId": "msp_carlos@example.com",
    "providerName": "ADFS2",
    "providerType": "SAML",
    "issuer": "http://auth2.example.com",
    "primary": false,
    "dateCreated": 111111111111111
}, {
    "userId": "msp_carlos@example.com",
    "providerName": "ADFS3",
    "providerType": "SAML",
    "issuer": "http://auth3.example.com",
    "primary": false,
    "dateCreated": 111111111111111
}]
```

Things to know about linking federated users

- You can link up to five federated users to each user profile.
- You can link users to each IdP from up to five IdP attribute claims, as defined by the `ProviderAttributeName` parameter of `SourceUser` in an `AdminLinkProviderForUser` API request. For example, if you have linked at least one user to the source attributes `email`, phone, `department`, `given_name`, and `location`, you can only link additional users on one of those five attributes.
- You can link federated users to either an existing federated user profile, or to a local user.
- You can't link providers to user profiles in the AWS Management Console.
- Your user's ID token contains all of their associated providers in the `identities` claim.
- You can set a password for the automatically-created federated user profile in an AdminSetUserPassword API request. That user's status then changes from `EXTERNAL_PROVIDER` to `CONFIRMED`. A user in this state can sign in as a federated user, and initiate authentication flows in the API like a linked local user. They can also modify their password and attributes in token-authenticated API requests like ChangePassword and UpdateUserAttributes. As a best security practice and to keep users in sync with your external IdP, don't set passwords on federated user profiles. Instead, link users to local profiles with `AdminLinkProviderForUser`.
- Amazon Cognito populates user attributes to a linked local user profile when the user signs in through their IdP. Amazon Cognito processes identity claims in the ID token from an OIDC IdP, and also checks the `userInfo` endpoint of both OAuth 2.0 and OIDC providers. Amazon Cognito prioritizes information in an ID token over information from `userInfo`.

When you learn that your user is no longer using an external user account that you've linked to their profile, you can disassociate that user account with your user pool user. When you linked your user, you supplied the user's attribute name, attribute value and provider name in the request. To remove a profile that your user no longer needs, make an AdminDisableProviderForUser API request with equivalent parameters.

See AdminLinkProviderForUser for additional command syntax and examples in the AWS SDKs.

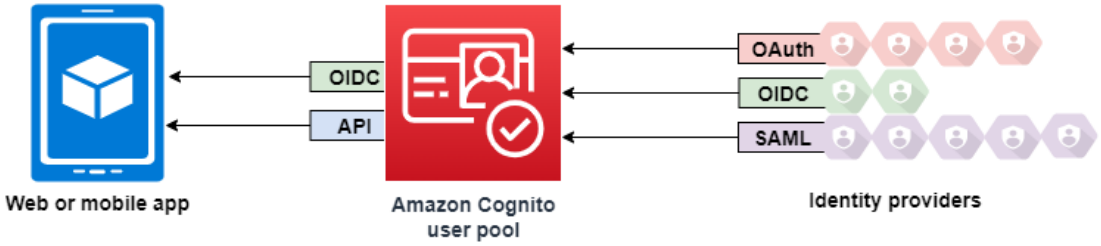
Accessing resources after successful sign-in

Your app users can either sign in directly through a user pool, or they can federate through a third-party identity provider (IdP). The user pool manages the overhead of handling the tokens that are returned from social sign-in through Facebook, Google, Amazon, and Apple, and from OpenID Connect (OIDC) and SAML IdPs. For more information, see Understanding user pool JSON web tokens (JWTs).

After a successful authentication, your app will receive user pool tokens from Amazon Cognito. You can use user pool tokens to:

- Retrieve AWS credentials that authorize requests for application resources in AWS services like Amazon DynamoDB and Amazon S3.
- Provide temporary, revocable proof of authentication.
- Populate identity data to a user profile in your app.
- Authorize changes to the signed-in user's profile in the user pool directory.
- Authorize requests for user information with an access token.
- Authorize requests to data that is behind access-protected external APIs with access tokens.
- Authorize access to application assets that are stored on the client or server with Amazon Verified Permissions.

For more information, see An example authentication session and Understanding user pool JSON web tokens (JWTs).



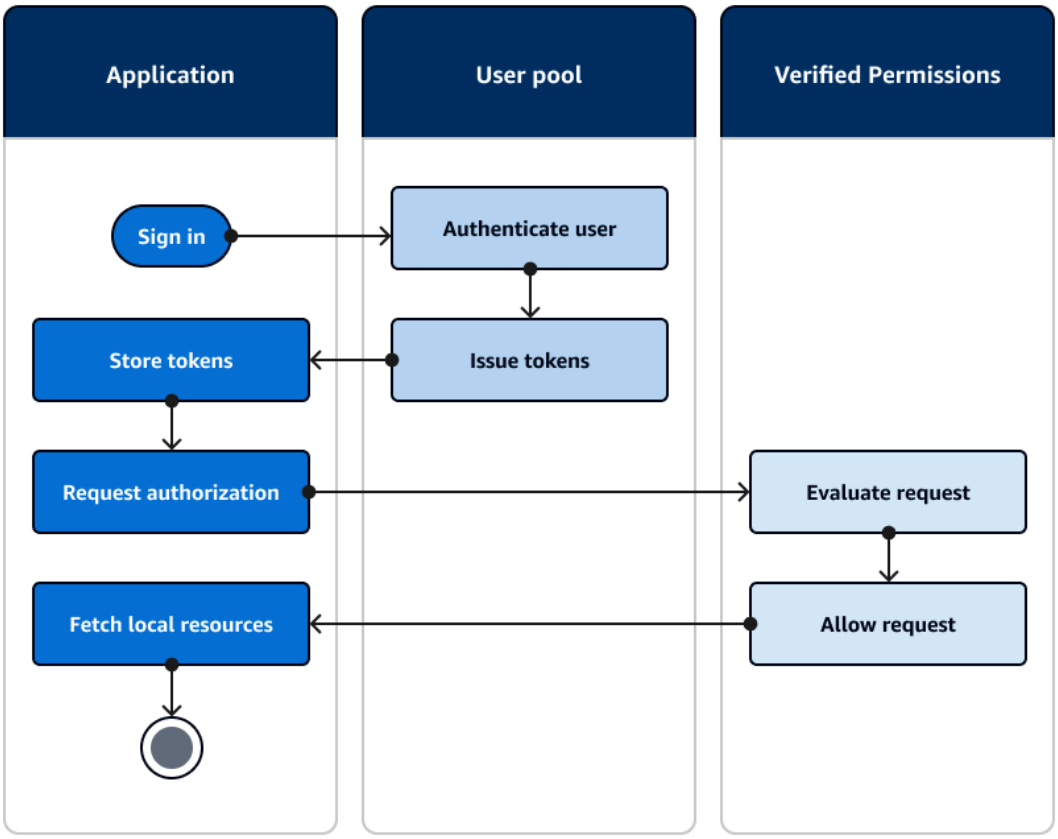
Topics

- Authorizing access to client or server resources with Amazon Verified Permissions
- Accessing resources with API Gateway after sign-in
- Accessing AWS services using an identity pool after sign-in

Authorizing access to client or server resources with Amazon Verified Permissions

Your app can pass the tokens from a signed-in user to Amazon Verified Permissions. Verified Permissions is a scalable, fine-grained permissions management and authorization service for applications that you've built. An Amazon Cognito user pool can be an identity source to a Verified Permissions policy store. Verified Permissions makes authorization decisions for requested actions and resources, like `GetPhoto` for `premium_badge.png`, from the principal and their attributes in user pool tokens.

The following diagram shows how your application can pass a user's token to Verified Permissions in an authorization request.



Get started with Amazon Verified Permissions

After you integrate your user pool with Verified Permissions, you gain a central source of granular authorization for all of your Amazon Cognito apps. This removes the need for fine-grained security logic that you would otherwise have to code and replicate between all of your apps. For more information about authorization with Verified Permissions, see Authorization with Amazon Verified Permissions.

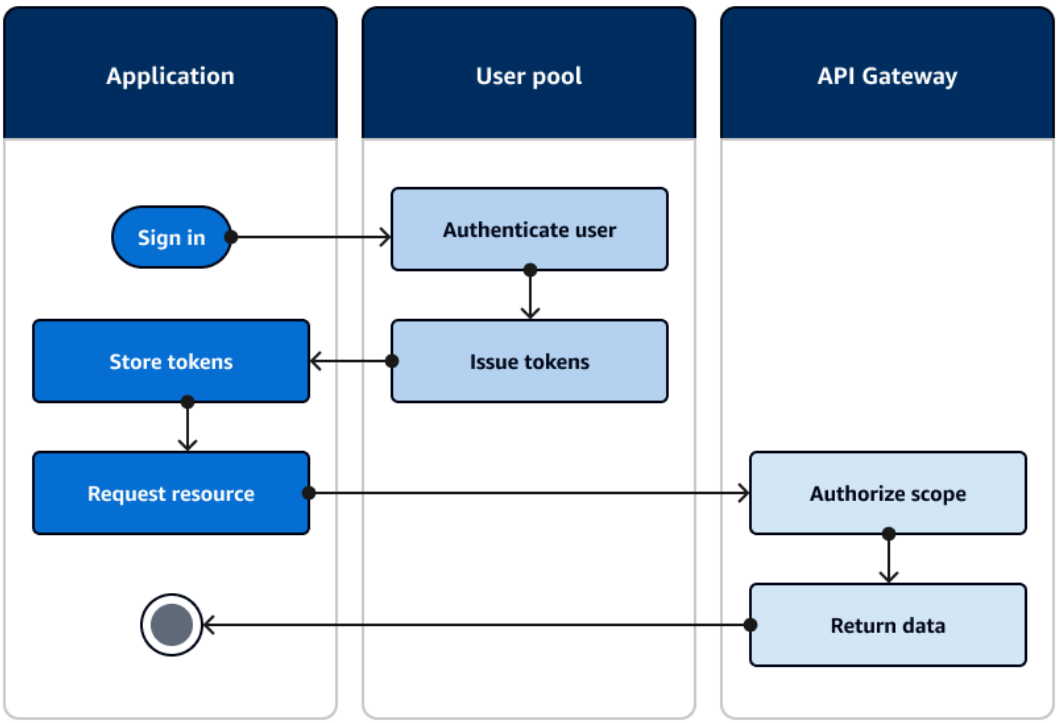
Verified Permissions authorization requests require AWS credentials. You can implement some of the following techniques to safely apply credentials to authorization requests.

- Operate a web application that can store secrets in the server backend.
- Acquire authenticated identity pool credentials.
- Proxy user requests through an access-token-authorized API, and append AWS credentials to the request.

Accessing resources with API Gateway after sign-in

A common use of Amazon Cognito user pools tokens is to authorize requests to an API Gateway REST API. The OAuth 2.0 scopes in access tokens can authorize a method and path, like HTTP GET for `/app_assets`. ID tokens can serve as generic authentication to an API and can pass user attributes to the backend service. API Gateway has additional custom authorization options like JWT authorizers for HTTP APIs and Lambda authorizers that can apply more fine-grained logic.

The following diagram illustrates an application that is gaining access to a REST API with the OAuth 2.0 scopes in an access token.



Your app must collect the tokens from authenticated sessions and add them as bearer tokens to an `Authorization` header in the request. Configure the authorizer that you configured for the API, path, and method to evaluate token contents. API Gateway returns data only if the request matches the conditions that you set up for your authorizer.

Some potential ways that API Gateway API can approve access from an application are:

- The access token is valid, isn't expired, and contains the correct OAuth 2.0 scope. The Amazon Cognito user pools authorizer for a REST API is a common implementation with a low barrier to

entry. You can also evaluate the body, query string parameters, and headers of a request to this type of authorizer.

- The ID token is valid and isn't expired. When you pass an ID token to an Amazon Cognito authorizer, you can perform additional validation of the ID token contents on your application server.
- A group, claim, attribute, or role in an access or ID token meets the requirements that you define in a Lambda function. A Lambda authorizer parses the token in the request header and evaluates it for an authorization decision. You can construct custom logic in your function or make an API request to Amazon Verified Permissions.

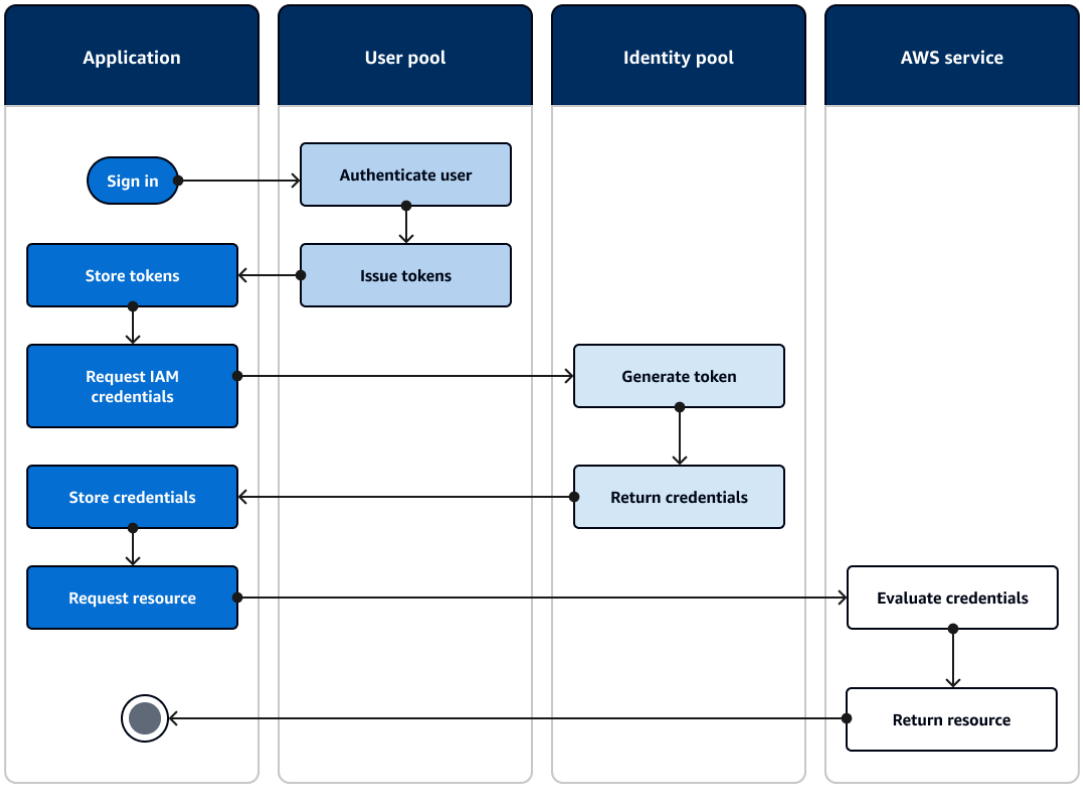
You can also authorize requests to an AWS AppSync GraphQL API with tokens from a user pool.

Accessing AWS services using an identity pool after sign-in

After your users sign in with a user pool, they can access AWS services with temporary API credentials that are issued from an identity pool.

Your web or mobile app receives tokens from a user pool. When you configure your user pool as an identity provider to your identity pool, the identity pool exchanges tokens for temporary AWS credentials. These credentials can be scoped to IAM roles and their policies that give users access to a limited set of AWS resources. For more information, see Identity pools authentication flow.

The following diagram shows how an application signs in with a user pool, retrieves identity pool credentials, and requests an asset from an AWS service.



You can use identity pool credentials to:

- Make fine-grained authorization requests to Amazon Verified Permissions with your user's own credentials.
- Connect to an Amazon API Gateway REST API or an AWS AppSync GraphQL API that authorizes connections with IAM.
- Connect to a database backend like Amazon DynamoDB or Amazon RDS that authorizes connections with IAM.
- Retrieve application assets from an Amazon S3 bucket.
- Initiate a session with an Amazon WorkSpaces virtual desktop.

Identity pools don't operate exclusively within an authenticated session with a user pool. They also accept authentication directly from third-party identity providers and can generate credentials for unauthenticated guest users.

For more information about using identity pools together with user pool groups to control access to your AWS resources, see Adding groups to a user pool and Using role-based access control. Also,

for more information about identity pools and AWS Identity and Access Management, see Identity pools authentication flow.

Setting up a user pool with the AWS Management Console

Create an Amazon Cognito user pool and make a note of the **User Pool ID** and **App Client ID** for each of your client apps. For more information about creating user pools, see Create a new application in the Amazon Cognito console.

Setting up an identity pool with the AWS Management Console

The following procedure describes how to use the AWS Management Console to integrate an identity pool with one or more user pools and client apps.

To add an Amazon Cognito user pools identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Amazon Cognito user pool**.
5. Enter a **User pool ID** and an **App client ID**.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

    a. You can give users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With an Amazon Cognito user pool IdP, you can also **Choose role with preferred_role claim in tokens**. For more information about the `cognito:preferred_role` claim, see Assigning precedence values to groups.

        i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to use to compare the claim to the rule, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

        ii. If you chose **Choose role with preferred_role claim in tokens**, Amazon Cognito issues credentials for the role in your user's `cognito:preferred_role` claim. If no

preferred role claim is present, Amazon Cognito issues credentials based on your **Role resolution**.

b. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

   - To apply no principal tags, choose **Inactive**.
   - To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.
   - To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Integrating a user pool with an identity pool

After your app user is authenticated, add that user's identity token to the logins map in the credentials provider. The provider name will depend on your Amazon Cognito user pool ID. It will have the following structure:

```
cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>
```

You can derive the value for *<region>* from the **User Pool ID**. For example, if the user pool ID is `us-east-1_EXAMPLE1`, then the *<region>* is `us-east-1`. If the user pool ID is `us-west-2_EXAMPLE2`, then the *<region>* is `us-west-2`.

JavaScript

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

var cognitoUser = userPool.getCurrentUser();

if (cognitoUser != null) {
 cognitoUser.getSession(function(err, result) {
  if (result) {
   console.log('You are now logged in.');

   // Create a credentials provider with the user's ID token in the logins map.
```

```
   const region = '<region>';
   const credentials = fromCognitoIdentityPool({
    identityPoolId: 'YOUR_IDENTITY_POOL_ID',
    logins: {
     'cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>':
 result.getIdToken().getJwtToken()
    },
    clientConfig: { region }
   });

   // Attach the provider to each service client that needs credentials.
   const client = new SomeServiceClient({ region, credentials });
  }
 });
}
```

Android

```
cognitoUser.getSessionInBackground(new AuthenticationHandler() {
 @Override
 public void onSuccess(CognitoUserSession session) {
  String idToken = session.getIdToken().getJWTToken();

  Map<String, String> logins = new HashMap<String, String>();
  logins.put("cognito-idp.<region>.amazonaws.com/<YOUR_USER_POOL_ID>",
 session.getIdToken().getJWTToken());
  credentialsProvider.setLogins(logins);
 }

});
```

iOS - objective-C

```
AWSServiceConfiguration *serviceConfiguration = [[AWSServiceConfiguration alloc]
 initWithRegion:AWSRegionUSEast1 credentialsProvider:nil];
AWSCognitoIdentityUserPoolConfiguration *userPoolConfiguration =
 [[AWSCognitoIdentityUserPoolConfiguration alloc] initWithClientId:@"YOUR_CLIENT_ID"
  clientSecret:@"YOUR_CLIENT_SECRET" poolId:@"YOUR_USER_POOL_ID"];
[AWSCognitoIdentityUserPool
 registerCognitoIdentityUserPoolWithConfiguration:serviceConfiguration
 userPoolConfiguration:userPoolConfiguration forKey:@"UserPool"];
AWSCognitoIdentityUserPool *pool = [AWSCognitoIdentityUserPool
 CognitoIdentityUserPoolForKey:@"UserPool"];
```

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc] initWithRegionType:AWSRegionUSEast1 identityPoolId:@"YOUR_IDENTITY_POOL_ID"
 identityProviderManager:pool];
```

iOS - swift

```
let serviceConfiguration = AWSServiceConfiguration(region: .USEast1,
 credentialsProvider: nil)
let userPoolConfiguration = AWSCognitoIdentityUserPoolConfiguration(clientId:
 "YOUR_CLIENT_ID", clientSecret: "YOUR_CLIENT_SECRET", poolId: "YOUR_USER_POOL_ID")
AWSCognitoIdentityUserPool.registerCognitoIdentityUserPoolWithConfiguration(serviceConfigura
 userPoolConfiguration: userPoolConfiguration, forKey: "UserPool")
let pool = AWSCognitoIdentityUserPool(forKey: "UserPool")
let credentialsProvider = AWSCognitoCredentialsProvider(regionType: .USEast1,
 identityPoolId: "YOUR_IDENTITY_POOL_ID", identityProviderManager:pool)
```

Understanding user pool JSON web tokens (JWTs)

Tokens are artifacts of authentication that your applications can use as proof of OIDC authentication and to request access to resources. The *claims* in tokens are information about your user. The ID token contains claims about their identity, like their username, family name, and email address. The access token contains claims like `scope` that the authenticated user can use to access third-party APIs, Amazon Cognito user self-service API operations, and the userInfo endpoint. The access and ID tokens both include a `cognito:groups` claim that contains your user's group membership in your user pool. For more information about user pool groups, see Adding groups to a user pool.

Amazon Cognito also has refresh tokens that you can use to get new tokens or revoke existing tokens. Refresh a token to retrieve a new ID and access tokens. Revoke a token to revoke user access that is allowed by refresh tokens.

Amazon Cognito issues tokens as base64url-encoded strings. You can decode any Amazon Cognito ID or access token from `base64url` to plaintext JSON. Amazon Cognito refresh tokens are encrypted, opaque to user pools users and administrators, and can only be read by your user pool.

Authenticating with tokens

When a user signs into your app, Amazon Cognito verifies the login information. If the login is successful, Amazon Cognito creates a session and returns an ID token, an access token, and a refresh token for the authenticated user. You can use the tokens to grant your users access to downstream resources and APIs like Amazon API Gateway. Or you can exchange them for temporary AWS credentials to access other AWS services.

Storing tokens

Your app must be able to store tokens of varying sizes. Token size can change for reasons including, but not limited to, additional claims, changes in encoding algorithms, and changes in encryption algorithms. When you enable token revocation in your user pool, Amazon Cognito adds additional

claims to JSON Web Tokens, increasing their size. The new claims `origin_jti` and `jti` are added to access and ID tokens. For more information about token revocation, see Revoking tokens.

> **Important**
>
> As a best practice, secure all tokens in transit and storage in the context of your application. Tokens can contain personally-identifying information about your users, and information about the security model that you use for your user pool.

Customizing tokens

You can customize the access and ID tokens that Amazon Cognito passes to your app. In a Pre token generation Lambda trigger, you can add, modify, and suppress token claims. The pre token generation trigger is a Lambda function that Amazon Cognito sends a default set of claims to. The claims include OAuth 2.0 scopes, user pool group membership, user attributes, and others. The function can then take the opportunity to make changes at runtime and return updated token claims to Amazon Cognito.

Additional costs apply to access token customization with version 2 events. For more information, see Amazon Cognito Pricing.

Topics

- Understanding the identity (ID) token
- Understanding the access token
- Parse user pool tokens safely
- Refresh tokens
- Ending user sessions with token revocation
- Verifying JSON web tokens
- Managing user pool token expiration and caching
- Scopes, M2M, and resource servers

Understanding the identity (ID) token

The ID token is a JSON Web Token (JWT) that contains claims about the identity of the authenticated user, such as `name`, `email`, and `phone_number`. You can use this identity

information inside your application. The ID token can also be used to authenticate users to your resource servers or server applications. You can also use an ID token outside of the application with your web API operations. In those cases, you must verify the signature of the ID token before you can trust any claims inside the ID token. See Verifying JSON web tokens.

You can set the ID token expiration to any value between 5 minutes and 1 day. You can set this value per app client.

The ID token includes the `acr` and `amr` claims, which describe the authentication level that the user satisfied and the methods that they completed. For more information, see ACR and AMR token claims.

> **Important**
>
> When your user signs in with managed login, Amazon Cognito sets session cookies that are valid for 1 hour. If you use managed login for authentication in your application, and specify a minimum duration of less than 1 hour for your access and ID tokens, your users will still have a valid session until the cookie expires. If the user has tokens that expire during the one-hour session, the user can refresh their tokens without the need to reauthenticate.

ID Token Header

The header contains two pieces of information: the key ID (`kid`), and the algorithm (`alg`).

```
{
"kid" : "1234example=",
"alg" : "RS256"
}
```

`kid`

The key ID. Its value indicates the key that was used to secure the JSON Web Signature (JWS) of the token. You can view your user pool signing key IDs at the `jwks_uri` endpoint.

For more information about the `kid` parameter, see the Key identifier (kid) header parameter.

`alg`

The cryptographic algorithm that Amazon Cognito used to secure the access token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256.

For more information about the `alg` parameter, see Algorithm (alg) header parameter.

ID token default payload

This is a example payload from an ID token. It contains claims about the authenticated user. For more information about OpenID Connect (OIDC) standard claims, see the list of OIDC standard claims. You can add claims of your own design with a Pre token generation Lambda trigger.

This payload is representative, not exhaustive. The set of claims in a Amazon Cognito ID token grows over time as new features add new claims, so the claims that your tokens carry can differ from this example. Treat this example as a snapshot of common claims rather than a complete, fixed list, and read How to parse this token safely before you write code that parses Amazon Cognito tokens.

```
<header>.{
    "sub": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "cognito:groups": [
        "test-group-a",
        "test-group-b",
        "test-group-c"
    ],
    "email_verified": true,
    "cognito:preferred_role": "arn:aws:iam::111122223333:role/my-test-role",
    "iss": "https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",
    "cognito:username": "my-test-user",
    "middle_name": "Jane",
    "nonce": "abcdefg",
    "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "cognito:roles": [
        "arn:aws:iam::111122223333:role/my-test-role"
    ],
    "aud": "xxxxxxxxxxxxexample",
    "identities": [
        {
            "userId": "amzn1.account.EXAMPLE",
            "providerName": "LoginWithAmazon",
            "providerType": "LoginWithAmazon",
```

```
            "issuer": null,
            "primary": "true",
            "dateCreated": "1642699117273"
        }
    ],
    "event_id": "64f513be-32db-42b0-b78e-b02127b4f463",
    "token_use": "id",
    "auth_time": 1676312777,
    "exp": 1676316377,
    "iat": 1676312777,
    "jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "email": "my-test-user@example.com"
}
.<token signature>
```

sub

A unique identifier (UUID), or subject, for the authenticated user. The username might not be unique in your user pool. The sub claim is the best way to identify a given user.

> **Important**
>
> Amazon Cognito generates sub in an Amazon Cognito-specific format that doesn't conform to a specific UUID format, including RFC UUID. You shouldn't strictly validate the format of sub. More generally, as described in How to parse this token safely, read each claim as the type that this page documents and don't impose a stricter format than the documentation states.

cognito:groups

An array of strings. Each string is the name of a user pool group that has your user as a member. Groups can be an identifier that you present to your app, or they can generate a request for a preferred IAM role from an identity pool. This claim is always an array, even when the user belongs to a single group.

cognito:preferred_role

The ARN of the IAM role that you associated with your user's highest-priority user pool group. For more information about how your user pool selects this role claim, see Assigning precedence values to groups.

`iss`

A single string. The issuer of the token. This claim identifies the user pool that generated the token. Your application should validate that this value matches your user pool's expected issuer URL. The claim has the following format.

`https://cognito-idp.<Region>.amazonaws.com/<your user pool ID>`

Your user pool can use an original or updated issuer. Updated issuers host the same JWKS content in multiple Regions, resulting in improved resilience and efficiency. For more information, see Amazon Cognito user pools as an OIDC issuer.

`cognito:username`

The username of your user in your user pool.

`nonce`

The `nonce` claim comes from a parameter of the same name that you can add to requests to your OAuth 2.0 `authorize` endpoint. When you add the parameter, the `nonce` claim is included in the ID token that Amazon Cognito issues, and you can use it to guard against replay attacks. If you do not provide a `nonce` value in your request, Amazon Cognito automatically generates and validates a nonce when you authenticate through a third-party identity provider, then adds it as a `nonce` claim to the ID token. The implementation of the `nonce` claim in Amazon Cognito is based on OIDC standards.

`origin_jti`

A token-revocation identifier associated with your user's refresh token. Amazon Cognito references the `origin_jti` claim when it checks if you revoked your user's token with the Revoke endpoint or the RevokeToken API operation. When you revoke a token, Amazon Cognito invalidates all access and ID tokens with the same `origin_jti` value.

`cognito:roles`

An array of strings. Each string is the ARN of an IAM role associated with one of your user's groups. Every user pool group can have one IAM role associated with it. This array represents all IAM roles for your user's groups, regardless of precedence, and is always an array even when it contains a single role. For more information, see Adding groups to a user pool.

`aud`

A single string. The user pool app client that authenticated your user. Amazon Cognito renders the same value in the access token `client_id` claim.

`identities`

An array of objects. Each object describes one third-party identity provider profile that you've linked to the user, either by federated sign-in or by linking a federated user to a local profile. Each object contains the fields `userId` (the user's unique ID at the provider), `providerName`, `providerType`, `issuer`, `primary`, and `dateCreated`. All of these fields are strings, with two serialization details to parse carefully: `primary` is the string `"true"` or `"false"`, not a JSON boolean, and `dateCreated` is a string that holds Unix time in milliseconds (for example `"1642699117273"`), not a number. The `issuer` field can be `null`. This claim is always an array, even when the user has one linked identity.

`token_use`

The intended purpose of the token. In an ID token, its value is `id`.

`auth_time`

A number. The authentication time, in Unix time format, when your user completed authentication.

`exp`

A number. The expiration time, in Unix time format, when your user's token expires.

`iat`

A number. The issued-at time, in Unix time format, when Amazon Cognito issued your user's token.

`jti`

The unique identifier of the JWT.

`email`

A single string. The email address of your user, when the `email` attribute is present and readable by the app client.

The ID token can contain OIDC standard claims that are defined in OIDC standard claims. The ID token can also contain custom attributes that you define in your user pool. Amazon Cognito writes custom attribute values to the ID token as strings regardless of attribute type.

> **Note**
>
> User pool custom attributes are always prefixed with `custom:`.

How to parse this token safely

The claim set in a Amazon Cognito ID token is not fixed. The claims that a token carries vary with the authentication flow, the user's group membership, and the identity provider. They also depend on the features that are enabled on your user pool. The OpenID Connect specification requires only `iss`, `sub`, `aud`, `exp`, and `iat`, and these are present in every Amazon Cognito ID token. Amazon Cognito also always adds `token_use` with the value `id`, which you should verify to confirm that you received an ID token rather than an access token. Treat every other claim as optional and conditional on how the user authenticated.

The ID token is one of several Amazon Cognito token types, and the same parsing rules apply to all of them. For more information about parsing tokens safely, see Parse user pool tokens safely.

Example ID tokens by scenario

The ID token default payload combines many optional claims into a single example. In practice, the claims that a token carries depend on how the user signed in. The following examples show the claim combinations that Amazon Cognito emits for four common scenarios. Each example is a decoded payload, not a signed and serialized JWT: a real token is `header.payload.signature`, base64url-encoded and RS256-signed with your user pool's key. The `sub`, user pool ID, and timestamp values are illustrative placeholders.

Baseline authorization-code ID token

The identity token that most applications consume after they exchange an authorization code at the token endpoint. The `token_use` value of `id` distinguishes it from an access token. The `nonce` claim is present because the client sent a `nonce` parameter in the authorization request to help prevent replay attacks.

```
{
    "sub": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
    "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ExAmPlE",
    "aud": "6o7h8i9jexampleclientid",
    "cognito:username": "my-test-user",
```

```
    "email": "my-test-user@example.com",
    "email_verified": true,
    "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "token_use": "id",
    "auth_time": 1759345200,
    "iat": 1759345200,
    "exp": 1759348800,
    "jti": "11112222-3333-4444-5555-666677778888",
    "nonce": "n-0S6_WzA2Mj"
}
```

Group and role membership

Amazon Cognito adds the group and role authorization claims when the user belongs to one or more user pool groups. The `cognito:groups` and `cognito:roles` claims are arrays; read them as arrays even when they contain a single element. `cognito:preferred_role` is a single string.

```
{
    "sub": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
    "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ExAmPlE",
    "aud": "6o7h8i9jexampleclientid",
    "cognito:username": "my-test-user",
    "cognito:groups": ["admins", "beta-testers"],
    "cognito:roles": ["arn:aws:iam::111122223333:role/my-test-role"],
    "cognito:preferred_role": "arn:aws:iam::111122223333:role/my-test-role",
    "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "token_use": "id",
    "auth_time": 1759345200,
    "iat": 1759345200,
    "exp": 1759348800,
    "jti": "11112222-3333-4444-5555-666677778888"
}
```

Federated sign-in

A user who signed in through an external identity provider, such as a social or SAML/OIDC provider, carries the `identities` claim, which describes the external provider link.

```
{
    "sub": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
    "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ExAmPlE",
    "aud": "6o7h8i9jexampleclientid",
```

```
    "cognito:username": "LoginWithAmazon_amzn1.account.EXAMPLE",
    "identities": [
        {
            "userId": "amzn1.account.EXAMPLE",
            "providerName": "LoginWithAmazon",
            "providerType": "LoginWithAmazon",
            "issuer": null,
            "primary": "true",
            "dateCreated": "1642699117273"
        }
    ],
    "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "token_use": "id",
    "auth_time": 1759345200,
    "iat": 1759345200,
    "exp": 1759348800,
    "jti": "11112222-3333-4444-5555-666677778888"
}
```

Note

Inside each `identities` object, `primary` and `dateCreated` are serialized as strings (`"true"` and `"1642699117273"`), not as a JSON boolean and number. A comparison such as `identity.primary === true` silently evaluates to false. Read `primary` as the string `"true"` or `"false"`, and `dateCreated` as a string that holds Unix time in milliseconds.

Authentication strength (ACR and AMR)

When your user pool reports the authentication level that the user reached, Amazon Cognito adds the `acr` and `amr` claims. The following token is from a user who authenticated with a password and a TOTP from an authenticator app, which is the highest level. Each token contains exactly one of the possible `acr` levels. For the fixed set of levels, the factor combinations that satisfy each one, and the complete list of `amr` values, see Authentication levels with ACR and AMR claims.

```
{
    "sub": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
    "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ExAmPlE",
    "aud": "6o7h8i9jexampleclientid",
    "cognito:username": "my-test-user",
    "acr": "urn:cognito:loa:4",
```

```
    "amr": ["pwd", "otp", "mfa"],
    "origin_jti": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
    "token_use": "id",
    "auth_time": 1759345200,
    "iat": 1759345200,
    "exp": 1759348800,
    "jti": "11112222-3333-4444-5555-666677778888"
}
```

Requesting an authentication level requires the Essentials or Plus feature plan. The `acr` and `amr` claims are computed by Amazon Cognito and can't be modified by a pre token generation Lambda trigger. Translating the Amazon Cognito levels to another standard, such as NIST or eIDAS, is the responsibility of your application.

ID Token Signature

The signature of the ID token is calculated based on the header and payload of the JWT token. Before you accept the claims in any ID token that your app receives, verify the signature of the token. For more information, see Verifying a JSON Web Token. Verifying JSON web tokens.

Understanding the access token

The user pool access token contains claims about the authenticated user, a list of the user's groups, and a list of scopes. The purpose of the access token is to authorize API operations. Your user pool accepts access tokens to authorize user self-service operations. For example, you can use the access token to grant your user access to add, change, or delete user attributes.

With OAuth 2.0 scopes in an access token, derived from the custom scopes that you add to your user pool, you can authorize your user to retrieve information from an API. For example, Amazon API Gateway supports authorization with Amazon Cognito access tokens. You can populate a REST API authorizer with information from your user pool. For details, see Control access to REST APIs using Amazon Cognito user pools as authorizer. You can also use Amazon Cognito as a JWT authorizer for an HTTP API. For details, see Control access to HTTP APIs with JWT authorizers in API Gateway. To generate an access token with custom scopes, you must request it through your user pool public endpoints.

With the Essentials or Plus feature plan, you can also implement a pre token generation Lambda trigger that adds scopes to your access tokens at runtime. For more information, see Pre token generation Lambda trigger.

A user's access token with the `openid` scope is permission to request more information about your user's attributes from the userInfo endpoint. The amount of information from the `userInfo` endpoint derives from the additional scopes in the access token: for example, `profile` for all user data, `email` for their email address.

A user's access token with the `aws.cognito.signin.user.admin` scope is permission to read and write user attributes, list authentication factors, configure multi-factor authentication (MFA) preferences, and manage remembered devices. The level of access to attributes that your access token grants to this scope matches the attribute read/write permissions you assign to your app client.

The access token is a JSON Web Token (JWT). The header for the access token has the same structure as the ID token. Amazon Cognito signs access tokens with a different key from the key that signs ID tokens. The value of an access key ID (`kid`) claim won't match the value of the `kid` claim in an ID token from the same user session. In your app code, verify ID tokens and access tokens independently. Don't trust the claims in an access token until you verify the signature. For more information, see Verifying JSON web tokens. You can set the access token expiration to any value between 5 minutes and 1 day. You can set this value per app client.

> **Important**
>
> For access and ID tokens, don't specify a minimum less than an hour if you use managed login. Managed login sets browsers cookies that are valid for one hour. If you configure an access token duration of less than an hour, this has no effect on the validity of the managed login cookie and users' ability to reauthenticate without additional credentials for one hour after initial sign-in.

The access token carries claims additively, the same way the ID token does, and the same parsing rules apply. For more information about parsing tokens safely, see Parse user pool tokens safely.

Access token header

The header contains two pieces of information: the key ID (`kid`), and the algorithm (`alg`).

```
{
"kid" : "1234example="
"alg" : "RS256",
```

```
}
```

`kid`

The key ID. Its value indicates the key that was used to secure the JSON Web Signature (JWS) of the token. You can view your user pool signing key IDs at the `jwks_uri` endpoint.

For more information about the `kid` parameter, see the Key identifier (kid) header parameter.

`alg`

The cryptographic algorithm that Amazon Cognito used to secure the access token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256.

For more information about the `alg` parameter, see Algorithm (alg) header parameter.

Access token default payload

This is a sample payload from an access token. For more information, see JWT claims. You can add claims of your own design with a Pre token generation Lambda trigger.

```
<header>.
{
   "sub":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
   "device_key": "aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
   "cognito:groups":[
      "testgroup"
   ],
   "iss":"https://cognito-idp.us-west-2.amazonaws.com/us-west-2_example",
   "version":2,
   "client_id":"xxxxxxxxxxxxexample",
   "aud": "https://api.example.com",
   "origin_jti":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
   "event_id":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
   "token_use":"access",
   "scope":"phone openid profile resourceserver.1/appclient2 email",
   "auth_time":1676313851,
   "exp":1676317451,
   "iat":1676313851,
   "jti":"aaaaaaaa-bbbb-cccc-dddd-eeeeeeeeeeee",
   "username":"my-test-user"
}
```

```
.<token signature>
```

sub

A unique identifier (UUID), or subject, for the authenticated user. The username might not be unique in your user pool. The sub claim is the best way to identify a given user.

> **Important**
>
> Amazon Cognito generates sub in an Amazon Cognito-specific format that doesn't conform to a specific UUID format, including RFC UUID. You shouldn't strictly validate the format of sub.

cognito:groups

An array of the names of user pool groups that have your user as a member.

iss

The issuer of the token. This claim identifies the user pool that generated the token. Your application should validate that this value matches your user pool's expected issuer URL. The claim has the following format.

`https://cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE`

Your user pool can use an original or updated issuer. Updated issuers host the same JWKS content in multiple Regions, resulting in improved resilience and efficiency. For more information, see Amazon Cognito user pools as an OIDC issuer.

client_id

The user pool app client that authenticated your user. Amazon Cognito renders the same value in the ID token aud claim.

aud

The URL of the API that the access token is intended to authorize for. Present only if your application requested a resource binding from your authorization server.

origin_jti

A token-revocation identifier associated with your user's refresh token. Amazon Cognito references the origin_jti claim when it checks if you revoked your user's token with the

Revoke endpoint or the RevokeToken API operation. When you revoke a token, Amazon Cognito no longer validates access and ID tokens with the same `origin_jti` value.

`token_use`

The intended purpose of the token. In an access token, its value is `access`.

`scope`

A list of OAuth 2.0 scopes issued to the signed-in user. Scopes define the access that the token provides to external APIs, user self-service operations, and user data on the `userInfo` endpoint. A token from the Token endpoint can contain any scopes that your app client supports. A token from Amazon Cognito API sign-in only contains the scope `aws.cognito.signin.user.admin`.

`auth_time`

The authentication time, in Unix time format, that your user completed authentication.

`exp`

The expiration time, in Unix time format, that your user's token expires.

`iat`

The issued-at time, in Unix time format, that Amazon Cognito issued your user's token.

`jti`

The unique identifier of the JWT.

`username`

The user's username in the user pool.

More resources

- How to customize access tokens in Amazon Cognito user pools

Machine-to-machine access token payload

Because a machine-to-machine (M2M) access token authorizes an application instead of a user, its payload differs from the user access token described previously. In M2M authorization, the `sub` claim contains the ID of the app client that requested the token, not a user identifier, and the `sub` and `client_id` claims have the same value. The token carries the custom scopes that authorize

requests to your resource server, along with standard claims such as `iss`, `token_use` (with a value of `access`), `exp`, `iat`, and `jti`.

Because M2M authorization involves no user and issues no refresh token, an M2M access token omits several claims that a user access token contains. Your resource server can't expect these claims:

- The `username` and `cognito:groups` claims, which identify an authenticated user and their group membership.
- The `device_key` claim, which identifies a remembered user device.
- The `origin_jti` claim, which links a token to a revocable refresh token that M2M authorization doesn't issue.

Access token signature

The signature of the access token, signed with the key advertised at the `.well-known/jwks.json` endpoint, validates the integrity of the token header and payload. When you use access tokens to authorize access to external APIs, always configure your API authorizer to verify this signature against the key that signed it. For more information, see Verifying JSON web tokens.

Parse user pool tokens safely

An Amazon Cognito token's claim set is not fixed. The claims that your token carries vary with the authentication flow, the user's group membership, and the identity provider, and with the features that you enable on your user pool. Amazon Cognito adds new claims additively over time: it introduces new keys, but it does not rename, change the type of, or remove the existing claims that this guide documents. These rules apply to both ID tokens and access tokens.

Don't write your own token parser. Use a JSON Web Token (JWT) library and a JSON parser that keep working as the token grows. The two rules that prevent almost every real-world break are the first two below: tolerate claims you don't recognize, and make the right assumptions about the type and cardinality of each claim you read.

Ignore claims that you don't recognize

> Read the claims your application needs and leave the rest alone. Don't reject, fail, or throw an error when a token carries a claim that isn't listed in this guide. New claims are added additively, and a token that contains an unlisted claim is still valid.

Read each claim in its documented type

A JSON parser identifies each value's primitive type from the JSON itself, so the risk is in the assumptions your code then makes about that value. Treat each claim as the type and cardinality this guide documents for it. A claim that is documented as an array is always an array, even when it currently holds a single value. Don't collapse it to a string. For example, `cognito:groups`, `cognito:roles`, and `identities` are always arrays. The same principle applies to claims defined by other specifications. For example, `amr`, when present, is an array of strings as defined in RFC 8176 on the IETF website, so read it in that type.

Don't over-validate claim formats

Validate what the specification for a claim requires, and no more. For example, `sub` is a unique string but is not guaranteed to be in any particular UUID format. Some values inside custom claims such as `identities` are serialized as strings rather than as JSON booleans or numbers, and opaque values aren't guaranteed to be any particular length. Imposing a stricter format than the documentation states causes valid tokens to fail.

Don't assume the number of claims or their order

The claims in a token can appear in any order, and the count can change between tokens and over time. Address each claim by name rather than by position, and parse the token as a general JSON object rather than a fixed structure.

Budget for growth in token size

Because the claim set grows, the overall size of the token grows too. Don't assume a fixed or maximum token size in buffers, column widths, headers, or cookies.

The `openid-configuration` discovery document follows the same additive, forward-compatible approach. For more information about parsing that document safely, see Prepare for changes to the discovery document.

Refresh tokens

You can use the refresh token to retrieve new ID and access tokens. By default, the refresh token expires 30 days after your application user signs into your user pool. When you create an application for your user pool, you can set the application's refresh token expiration to any value between 60 minutes and 10 years.

Getting new access and identity tokens with a refresh token

Amazon Cognito issues refresh tokens in response to successful authentication with the managed login authorization-code flow and with API operations or SDK methods. The refresh token returns new ID and access tokens, and optionally a new refresh token. You can use refresh tokens in the following ways.

GetTokensFromRefreshToken

The GetTokensFromRefreshToken API operation issues new ID and access tokens from a valid refresh token. You also get a new refresh token if you've enabled refresh token rotation.

InitiateAuth and AdminitiateAuth

The AdminInitiateAuth or InitiateAuth API operations include the `REFRESH_TOKEN_AUTH` authentication flow. In this flow, you pass a refresh token and get new ID and access tokens. You can't authenticate with `REFRESH_TOKEN_AUTH` in app clients with refresh token rotation enabled.

OAuth token endpoint

The token endpoint in user pools with a domain has a `refresh_token` grant type that issues new ID, access, and optionally (with refresh token rotation) refresh tokens from a valid refresh token.

Refresh token rotation

You can optionally configure refresh token rotation in your app client. With refresh token rotation, your client can invalidate the original refresh token and issue a new refresh token with each token refresh. When this setting is enabled, each successful request in all forms of token refresh return a new ID, access, *and* refresh token. When this setting is disabled, token-refresh requests return new access and ID tokens only and the original refresh token remains valid. The new refresh token is valid for the remaining duration of the original refresh token. You can configure app clients to rotate refresh tokens or to carry over the original refresh token. To allow for retries for a brief duration, you can also configure a grace period for the original refresh token of up to 60 seconds.

> **Note**
>
> Rotation reissues a refresh token within the *same* fixed validity window. It doesn't extend the window. Each new refresh token is valid only for the time remaining on the original

`RefreshTokenValidity` period, no matter how many times the token rotates. For example, suppose you set `RefreshTokenValidity` to 30 days and a session rotates its refresh token every day. The session still expires 30 days after the *first* sign-in, not 30 days after the most recent rotation. To keep a user signed in past that window, they must reauthenticate.

Things to know about refresh token rotation

- After you enable refresh token rotation, new claims are added in JSON web tokens from your user pool. The `origin_jti` and `jti` claims are added to access and ID tokens. These claims increase the size of the JWTs.
- Refresh token rotation isn't compatible with the authentication flow `REFRESH_TOKEN_AUTH`. To implement refresh token rotation, you must disable this authentication flow in your app client and design your application to submit token-refresh requests with the GetTokensFromRefreshToken API operation or the equivalent SDK method.
- With refresh token rotation inactive, you can complete token-refresh requests with either `GetTokensFromRefreshToken` or `REFRESH_TOKEN_AUTH`.
- When device remembering is active in your user pool, you must provide the device key in `GetTokensFromRefreshToken` requests. If your user doesn't have a confirmed-device key that your application submits in the initial authentication request, Amazon Cognito issues a new one. To refresh tokens in this configuration, you must provide a device key, whether you specified one in `AuthParameters` or received a new one in the authentication response.
- You can pass `ClientMetadata` to the pre token generation Lambda trigger in your `GetTokensFromRefreshToken` request. This data, which gets passed to the input event for your trigger, delivers additional context that you can use in the custom logic of your Lambda function.

As a security best practice, enable refresh token rotation on your app clients.

Enable refresh token rotation (console)

The following procedure turns refresh token rotation on or off for your app client. This procedure requires an existing app client. To learn more about creating an app client, see Application-specific settings with app clients.

To enable refresh token rotation

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Navigate to the **App clients** menu and select an existing app client.
5. Select **Edit** from the **App client information** section of the page.
6. Under **Advanced security configurations**, locate the **Enable refresh token rotation** option.
7. To enable rotation, select the checkbox. To disable rotation, deselect the checkbox.
8. Under **Refresh token rotation grace period**, enter a number of seconds, up to 60, that you want to set as the delay before the rotated-out refresh token is revoked.

Enable refresh token rotation (API)

Configure refresh token rotation in a CreateUserPoolClient or UpdateUserPoolClient API request. The following partial request body turns on refresh token rotation and sets the grace period to ten seconds.

```
"RefreshTokenRotation" : {
   "Feature" : "ENABLED",
   "RetryGracePeriodSeconds" : 10
}
```

API and SDK token refresh

There are two ways to use the refresh token to get new ID and access tokens with the user pools API, depending on whether refresh token rotation is active. In app clients with refresh token rotation active, use the GetTokensFromRefreshToken API operation. In app clients without refresh token rotation, use the `REFRESH_TOKEN_AUTH` flow of the AdminInitiateAuth or InitiateAuth API operations.

> **Note**
>
> Users can authenticate with user pools in managed login or in custom applications that you build with AWS SDKs and Amazon Cognito API operations. The `REFRESH_TOKEN_AUTH` flow and `GetTokensFromRefreshToken` can both complete token refresh for

managed login users. Token refresh in custom applications doesn't affect managed login sessions. These sessions are set in a browser cookie and are valid for one hour. The `GetTokensFromRefreshToken` response issues new ID, access, and optionally refresh tokens, but doesn't renew the managed login session cookie.
`REFRESH_TOKEN_AUTH` isn't available in app clients with refresh token rotation enabled.

GetTokensFromRefreshToken

GetTokensFromRefreshToken returns new ID, access and refresh tokens from a request that you authorize with a refresh token. The following is an example request body for `GetTokensFromRefreshToken`. You can submit client metadata to Lambda triggers in requests to this operation.

```
{
    "RefreshToken": "eyJjd123abcEXAMPLE",
    "ClientId": "1example23456789",
    "ClientSecret": "myappclientsecret123abc",
    "ClientMetadata": {
      "MyMetadataKey" : "MyMetadataValue"
   },
}
```

AdminInitiateAuth/InitiateAuth

To use the refresh token when refresh token rotation is inactive, use the AdminInitiateAuth or InitiateAuth API operations. Pass `REFRESH_TOKEN_AUTH` for the `AuthFlow` parameter. In the `AuthParameters` property of `AuthFlow`, pass your user's refresh token as the value of `"REFRESH_TOKEN"`. Amazon Cognito returns new ID and access tokens after your API request passes all challenges.

The following is an example request body for a token refresh with the `InitiateAuth` or `AdminInitiateAuth` API.

```
{
    "AuthFlow": "REFRESH_TOKEN_AUTH",
    "ClientId": "1example23456789",
    "UserPoolId": "us-west-2_EXAMPLE",
    "AuthParameters": {
        "REFRESH_TOKEN": "eyJjd123abcEXAMPLE",
        "SECRET_HASH": "kT5acwCVrbD6JexhW3EQwnRSe6fLuPTRkEQ50athqv8="
```

```
    }
}
```

OAuth token refresh

You can also submit refresh tokens to the Token endpoint in a user pool where you have configured a domain. In the request body, include a `grant_type` value of `refresh_token` and a `refresh_token` value of your user's refresh token.

Requests to the token endpoint are available in app clients with refresh token rotation active and those where it's inactive. When refresh token rotation is active, the token endpoint returns a new refresh token.

The following is an example request with a refresh token.

```
POST /oauth2/token HTTP/1.1
Host: auth.example.com
Content-Type: application/x-www-form-urlencoded
Authorization: Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw
Content-Length: **

client_id=1example23456789&grant_type=refresh_token&refresh_token=eyJjd123abcEXAMPLE
```

Revoking refresh tokens

You can revoke refresh tokens that belong to a user. For more information about revoking tokens, see Ending user sessions with token revocation.

> **Note**
>
> Revoking the refresh token will revoke all ID and access tokens that Amazon Cognito issued from refresh requests with that token.

To sign users out from all current signed-in session, revoke all of their tokens with GlobalSignOut or AdminUserGlobalSignOut API requests. After the user is signed out, the following effects happen.

- The user's refresh token can't get new tokens for the user.

- The user's access token can't make token-authorized API requests.
- The user must re-authenticate to get new tokens. Because managed login session cookies don't expire automatically, your user can re-authenticate with a session cookie, with no additional prompt for credentials. After you sign out your managed login users, redirect them to the Logout endpoint, where Amazon Cognito clears their session cookie.

With refresh tokens, you can persist users' sessions in your app for a long time. Over time, your users might want to deauthorize some applications where they have stayed signed in with their refresh tokens. To sign your user out from a single session, revoke their refresh token. When your user wants to sign themself out from all authenticated sessions, generate a GlobalSignOut API request . Your app can present your user with a choice like **Sign out from all devices**. `GlobalSignOut` accepts a user's valid–unaltered, unexpired, not-revoked–access token. Because this API is token-authorized, one user can't use it to initiate sign-out for another user.

You can, however, generate an AdminUserGlobalSignOut API request that you authorize with your AWS credentials to sign out any user from all of their devices. The administrator application must call this API operation with AWS developer credentials and pass the user pool ID and the user's username as parameters. The `AdminUserGlobalSignOut` API can sign out any user in the user pool.

For more information about requests that you can authorize with either AWS credentials or a user's access token, see List of API operations grouped by authorization model.

Ending user sessions with token revocation

You can revoke refresh tokens and end user sessions with the following methods. When you revoke a refresh token, all access tokens that were previously issued by that refresh token become invalid. The other refresh tokens issued to the user are not affected.

RevokeToken operation

RevokeToken revokes all access tokens for a given refresh token, including the initial access token from interactive sign-in. This operation doesn't affect any of the user's other refresh tokens or the ID- and access-token children of those other refresh tokens.

Revocation endpoint

The revoke endpoint revokes a given refresh token and all ID and access tokens that the refresh token generated. This endpoint also revokes the initial access token from interactive sign-in.

Requests to this endpoint don't affect any of the user's other refresh tokens or the ID- and access-token children of those other refresh tokens.

GlobalSignOut operation

GlobalSignOut is a self-service operation that a user authorizes with their access token. This operation revokes all of the requesting user's refresh, ID, and access tokens.

AdminUserGlobalSignOut operation

AdminUserGlobalSignOut is a server-side operation that an administrator authorizes with IAM credentials. This operation revokes all of the target user's refresh, ID, and access tokens.

Things to know about revoking tokens

- Your request to revoke a refresh token must include the client ID that was used to obtain the token.
- User pool JWTs are self-contained with a signature and expiration time that was assigned when the token was created. Revoked tokens can't be used with any Amazon Cognito API calls that require a token. However, revoked tokens will still be valid if they are verified using any JWT library that verifies the signature and expiration of the token.
- When you create a new user pool client, token revocation is enabled by default.
- You can revoke refresh tokens only in app clients with token revocation enabled.
- After you enable token revocation, new claims are added in the Amazon Cognito JSON Web Tokens. The `origin_jti` and `jti` claims are added to access and ID tokens. These claims increase the size of the application client access and ID tokens.
- When you disable token revocation in an app client where it was previously enabled, revoked tokens don't become active again.
- When you disable a user account (which revokes refresh and access tokens), the revoked tokens don't become active if you enable the user account again.
- When you create a new user pool client using the AWS Management Console, the AWS CLI, or the AWS API, token revocation is enabled by default.

Enable token revocation

Before you can revoke a token for an existing user pool client, you must enable token revocation. You can enable token revocation for existing user pool clients using the AWS CLI or the AWS

API. To do this, call the `aws cognito-idp describe-user-pool-client` CLI command or the `DescribeUserPoolClient` API operation to retrieve the current settings from your app client. Then call the `aws cognito-idp update-user-pool-client` CLI command or the `UpdateUserPoolClient` API operation. Include the current settings from your app client and set the `EnableTokenRevocation` parameter to `true`.

To create or modify an app client with token revocation enabled with the Amazon Cognito API or with an AWS SDK, include the following parameter in your CreateUserPoolClient or UpdateUserPoolClient API request.

```
"EnableTokenRevocation": true
```

To configure token revocation in the Amazon Cognito console, select an app client from the **App clients** menu in your user pool. Select the **Edit** button in **App client information** and enable or disable token revocation under **Advanced configuration**.

Revoke a token

You can revoke a refresh token using a RevokeToken API request, for example with the `aws cognito-idp revoke-token` CLI command. You can also revoke tokens using the Revoke endpoint. This endpoint is available after you add a domain to your user pool. You can use the revocation endpoint on either an Amazon Cognito hosted domain or your own custom domain.

The following is the body of an example `RevokeToken` API request.

```
{
   "ClientId": "1example23456789",
   "ClientSecret": "abcdef123456789ghijklexample",
   "Token": "eyJjdHkiOiJKV1QiEXAMPLE"
}
```

The following is an example cURL request to the `/oauth2/revoke` endpoint of a user pool with a custom domain.

```
curl --location 'auth.mydomain.com/oauth2/revoke' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--header 'Authorization: Basic Base64Encode(client_id:client_secret)' \
--data-urlencode 'token=abcdef123456789ghijklexample' \
--data-urlencode 'client_id=1example23456789'
```

The `RevokeToken` operation and the `/oauth2/revoke` endpoint require no additional authorization unless your app client has a client secret.

Verifying JSON web tokens

JSON web tokens (JWTs) can be decoded, read, and modified easily. A modified access token creates a risk of privilege escalation. A modified ID token creates a risk of impersonation. Your application trusts your user pool as a token issuer, but what if a user intercepts the token in transit? You must ensure that your application is receiving the same token that Amazon Cognito issued.

Amazon Cognito issues tokens that use some of the integrity and confidentiality features of the OpenID Connect (OIDC) specification. User pool tokens indicate validity with objects like the expiration time, issuer, and digital signature. The signature, the third and final segment of the `.`-delimited JWT, is the key component of token validation. A malicious user can modify a token, but if your application retrieves the public key and compares the signature, it won't match. Any application that processes JWTs from OIDC authentication must perform this verification operation with each sign-in.

On this page, we make some general and specific recommendations for verification of JWTs. Application development spans a variety of programming languages and platforms. Because Amazon Cognito implements OIDC sufficiently close to the public specification, any reputable JWT library in your developer environment of choice can handle your verification requirements.

These steps describe verifying a user pool JSON Web Token (JWT).

Topics

- Prerequisites
- Validate tokens with aws-jwt-verify
- Understanding and inspecting tokens

Prerequisites

Your library, SDK, or software framework might already handle the tasks in this section. AWS SDKs provide tools for Amazon Cognito user pool token handling and management in your app. AWS Amplify includes functions to retrieve and refresh Amazon Cognito tokens.

For more information, see the following pages.

- Integrating Amazon Cognito authentication and authorization with web and mobile apps
- Code examples for Amazon Cognito Identity Provider using AWS SDKs
- Advanced workflows in the *Amplify Dev Center*

Many libraries are available for decoding and verifying a JSON Web Token (JWT). If you want to manually process tokens for server-side API processing, or if you are using other programming languages, these libraries can help. See the OpenID foundation list of libraries for working with JWT tokens.

Validate tokens with aws-jwt-verify

In a Node.js app, AWS recommends the aws-jwt-verify library to validate the parameters in the token that your user passes to your app. With `aws-jwt-verify`, you can populate a `CognitoJwtVerifier` with the claim values that you want to verify for one or more user pools. Some of the values that it can check include the following.

- That access or ID tokens aren't malformed or expired, and have a valid signature.
- That access tokens came from the correct user pools and app clients.
- That access token claims contain the correct OAuth 2.0 scopes.
- That the keys that signed your access and ID tokens match a signing key `kid` from the *JWKS URI* of your user pools.

  The JWKS URI contains public information about the private key that signed your user's token. You can find the JWKS URI for your user pool at `https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/.well-known/jwks.json`.

For more information and example code that you can use in a Node.js app or a AWS Lambda authorizer, see aws-jwt-verify on GitHub.

> **Note**
>
> When the `iss` (issuer) claim in a token doesn't match the user pool that you configured your `CognitoJwtVerifier` with, verification fails with a `ParameterValidationError`. This commonly happens when a token is presented to a verifier for a different user pool or AWS Region. Confirm that the `userPoolId` you pass to

`CognitoJwtVerifier.create()` matches the issuer of the tokens that your application receives.

Understanding and inspecting tokens

Before you integrate token inspection with your app, consider how Amazon Cognito assembles JWTs. Retrieve example tokens from your user pool. Decode and examine them in detail to understand their characteristics, and determine what you want to verify and when. For example, you might want to examine group membership in one scenario, and scopes in another.

The following sections describe a process to manually inspect Amazon Cognito JWTs as you prepare your app.

Confirm the structure of the JWT

A JSON Web Token (JWT) includes three sections with a `.` (dot) delimiter between them.

Header

The key ID, `kid`, and the RSA algorithm, `alg`, that Amazon Cognito used to sign the token. Amazon Cognito signs tokens with an `alg` of `RS256`. The `kid` is a truncated reference to a 2048-bit RSA private signing key held by your user pool.

Payload

Token claims. In an ID token, the claims include user attributes and information about the user pool, `iss`, and app client, `aud`. In an access token, the payload includes scopes, group membership, your user pool as `iss`, and your app client as `client_id`.

Signature

The signature isn't decodable base64url like the header and payload. It's an RSA256 identifier derived from a signing key and parameters that you can observe at your JWKS URI.

The header and payload are base64url-encoded JSON. You can identify them by the opening characters `eyJ` that decode to the starting character `{`. If your user presents a base64url-encoded JWT to your app and it's not in the format `[JSON Header].[JSON Payload].[Signature]`, it's not a valid Amazon Cognito token and you can discard it.

The following example application verifies user pool tokens with `aws-jwt-verify`.

```
// cognito-verify.js
// Usage example: node cognito-verify.js eyJra789ghiEXAMPLE

const { CognitoJwtVerifier } = require('aws-jwt-verify');

// Replace with your Amazon Cognito user pool ID
const userPoolId = 'us-west-2_EXAMPLE';

async function verifyJWT(token) {
  try {
    const verifier = CognitoJwtVerifier.create({
      userPoolId,
      tokenUse: 'access', // or 'id' for ID tokens
      clientId: '1example23456789', // you must verify the token audience
    });

    const payload = await verifier.verify(token);
    console.log('Decoded JWT:', payload);
  } catch (err) {
    console.error('Error verifying JWT:', err);
  }
}

// Example usage
if (process.argv.length < 3) {
  console.error('Please provide a JWT token as an argument.');
  process.exit(1);
}

const MyToken = process.argv[2];
verifyJWT(MyToken);
```

Validate the JWT

The JWT signature is a hashed combination of the header and the payload. Amazon Cognito generates two pairs of RSA cryptographic keys for each user pool. One private key signs access tokens, and the other signs ID tokens.

To verify the signature of a JWT token

1. Decode the ID token.

The OpenID Foundation also maintains a list of libraries for working with JWT tokens.

You can also use AWS Lambda to decode user pool JWTs. For more information, see Decode and verify Amazon Cognito JWT tokens using AWS Lambda.

2. Compare the local key ID (`kid`) to the public `kid`.

   a. Download and store the corresponding public JSON Web Key (JWK) for your user pool. It is available as part of a JSON Web Key Set (JWKS). You can locate it by constructing the following `jwks_uri` URI for your environment:

   ```
   https://cognito-idp.<Region>.amazonaws.com/<userPoolId>/.well-known/jwks.json
   ```

   For more information on JWK and JWK sets, see JSON Web Key (JWK).

   > **Note**
   >
   > Amazon Cognito might rotate signing keys in your user pool. As a best practice, cache public keys in your app, using the `kid` as a cache key, and refresh the cache periodically. Compare the `kid` in the tokens that your app receives to your cache. If you receive a token with the correct issuer but a different `kid`, Amazon Cognito might have rotated the signing key. Refresh the cache from your user pool `jwks_uri` endpoint.

   This is a sample `jwks.json` file:

   ```
   {
    "keys": [{
     "kid": "1234example=",
     "alg": "RS256",
     "kty": "RSA",
     "e": "AQAB",
     "n": "1234567890",
     "use": "sig"
    }, {
     "kid": "5678example=",
     "alg": "RS256",
     "kty": "RSA",
     "e": "AQAB",
   ```

```
  "n": "987654321",
  "use": "sig"
 }]
}
```

Key ID (`kid`)

The `kid` is a hint that indicates which key was used to secure the JSON Web Signature (JWS) of the token.

Algorithm (`alg`)

The `alg` header parameter represents the cryptographic algorithm that is used to secure the ID token. User pools use an RS256 cryptographic algorithm, which is an RSA signature with SHA-256. For more information on RSA, see RSA cryptography.

Key type (`kty`)

The `kty` parameter identifies the cryptographic algorithm family that is used with the key, such as "RSA" in this example.

RSA exponent (e)

The `e` parameter contains the exponent value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

RSA modulus (n)

The `n` parameter contains the modulus value for the RSA public key. It is represented as a Base64urlUInt-encoded value.

Use (`use`)

The `use` parameter describes the intended use of the public key. For this example, the `use` value `sig` represents signature.

b. Search the public JSON Web Key for a `kid` that matches the `kid` of your JWT.

Verify the claims

To verify JWT claims

1. Verify that the token hasn't expired. Decode the token and compare the `exp` claim to the current time. Because user pool JWTs are self-contained and carry their own expiration time, you can check expiration offline, without a network call to Amazon Cognito.

   > **Note**
   >
   > API operations such as GetUser or the userInfo endpoint endpoint return an error for an expired access token, but a rejected request is a side effect of authorization, not a token-verification method. Don't use these network calls to check expiration: decode the token and compare the `exp` claim instead.

2. The `aud` claim in an ID token and the `client_id` claim in an access token must match the app client ID that was created in the Amazon Cognito user pool.
3. The issuer (`iss`) claim must match your user pool. For example, a user pool created in the `us-east-1` Region will have the following `iss` value:

   `https://cognito-idp.us-east-1.amazonaws.com/<userpoolID>`.
4. Check the `token_use` claim.

   - If you are only accepting the access token in your web API operations, its value must be `access`.
   - If you are only using the ID token, its value must be `id`.
   - If you are using both ID and access tokens, the `token_use` claim must be either `id` or `access`.
5. Confirm that the token hasn't been revoked. A token with a valid signature and an unexpired `exp` claim can still have been revoked before its expiration time–for example, when a user signs out or when you call the RevokeToken API operation. Offline signature and expiration checks can't detect revocation on their own. If your application must honor revocation before a token expires, verify the token's status against Amazon Cognito–for example, with an authorized API request–or keep access token lifetimes short. For more information, see Ending user sessions with token revocation.

You can now trust the claims inside the token.

Note

Some AWS services can verify Amazon Cognito JWTs for you, rather than in your application code. If you use Amazon Cognito as a JWT authorizer for an Amazon API Gateway HTTP API, you provide the same issuer and audience values in the authorizer configuration. Set the issuer to your user pool endpoint, `https://cognito-idp.<Region>.amazonaws.com/<userPoolId>`, which is also the value of the `issuer` field in your user pool `/.well-known/openid-configuration` document. Set the audience to your app client ID.

API Gateway matches the audience against the token's `aud` claim, or against the `client_id` claim when the token has no `aud` claim. Amazon Cognito ID tokens carry `aud`, and access tokens carry `client_id` and no `aud` unless the request includes a resource-server binding. For an ID token, or an access token with no resource binding, set the audience to your app client ID. When your application requests an access token that is bound to a resource server, the token's `aud` claim is set to the resource identifier (the resource server URL) instead of the app client ID; in that case, configure the resource server URL as the audience so that API Gateway matches on it. For the API Gateway configuration steps, see Control access to REST APIs using Amazon Cognito user pools as authorizer and Control access to HTTP APIs with JWT authorizers in API Gateway.

An Application Load Balancer can also authenticate users with your user pool. The load balancer passes signed user claims to your targets in a JWT-formatted `x-amzn-oidc-data` HTTP header that your application verifies. For more information, see Authenticate users using an Application Load Balancer.

Managing user pool token expiration and caching

Your app must successfully complete one of the following requests each time you want to get a new JSON Web Token (JWT).

- Request a client credentials or authorization code grant from the Token endpoint.
- Request a client-credentials access token with the GetClientToken API operation, which requires no user pool domain.
- Request an implicit grant from your managed login pages.
- Authenticate a local user in an Amazon Cognito API request like InitiateAuth.

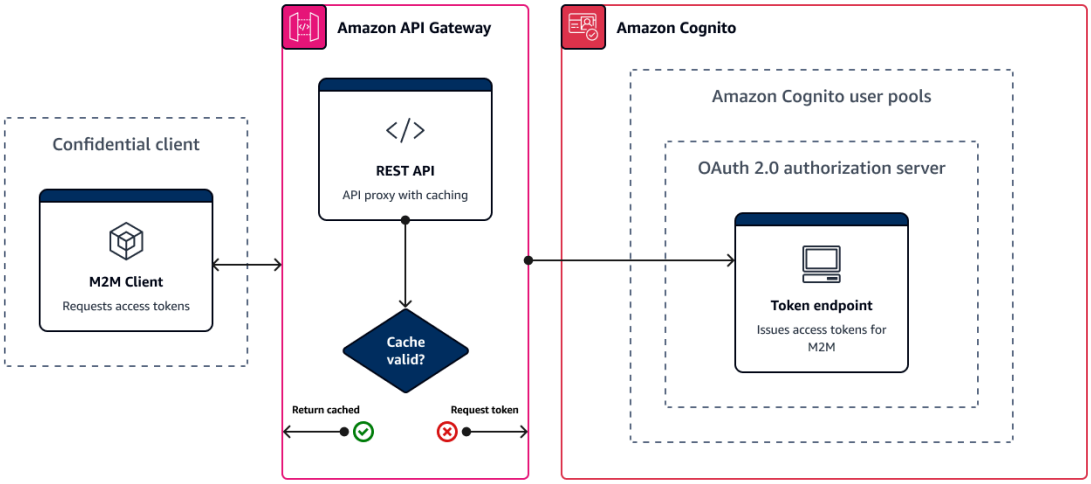
You can configure your user pool to set tokens to expire in minutes, hours, or days. To ensure the performance and availability of your app, use Amazon Cognito tokens for about 75% of the token lifetime, and only then retrieve new tokens. A cache solution that you build for your app keeps tokens available, and prevents the rejection of requests by Amazon Cognito when your request rate is too high. A client-side app must store tokens in a memory cache. A server-side app can add an encrypted cache mechanism to store tokens.

When your user pool generates a high volume of user or machine-to-machine activity, you might encounter the limits that Amazon Cognito sets on the number of requests for tokens that you can make. To reduce the number of requests you make to Amazon Cognito endpoints, you can either securely store and reuse authentication data, or implement exponential backoff and retries.

Authentication data comes from two classes of endpoints. Amazon Cognito OAuth 2.0 endpoints include the token endpoint, which services client credentials and managed login authorization code requests. Service endpoints answer user pools API requests like `InitiateAuth` and `RespondToAuthChallenge`. Each type of request has its own limit. For more information about limits, see Quotas in Amazon Cognito.

Caching machine-to-machine access tokens with Amazon API Gateway

With API Gateway token caching, your app can scale in response to events larger than the default request rate quota of Amazon Cognito OAuth endpoints.



You can cache the access tokens so that your app only requests a new access token if a cached token is expired. Otherwise, your caching endpoint returns a token from the cache. This prevents an additional call to an Amazon Cognito API endpoint. When you use Amazon API Gateway as a

proxy to the Token endpoint, your API responds to the majority of requests that would otherwise contribute to your request quota, avoiding unsuccessful requests as a result of rate limiting.

The following API Gateway-based solution offers a low-latency, low-code/no-code implementation of token caching. API Gateway APIs are encrypted in transit, and optionally at rest. An API Gateway cache is ideal for the OAuth 2.0 client credentials grant, a frequently high-volume grant type that produces access tokens to authorize machine-to-machine and microservice sessions. In an event like a traffic surge that causes your microservices to horizontally scale, you can end up with many systems using the same client credentials at a volume that exceeds the AWS request-rate limit of your user pool or app client. To preserve app availability and low latency, a caching solution is best practice in such scenarios.

In this solution, you define a cache in your API to store a separate access token for each combination of OAuth scopes and app client that you want to request in your app. When your app makes a request that matches the cache key, your API responds with an access token that Amazon Cognito issued to the first request that matched the cache key. When your cache key duration expires, your API forwards the request to your token endpoint and caches a new access token.

> **Note**
>
> Your cache key duration must be shorter than the access token duration of your app client.

The cache key is a combination of the OAuth scopes that you request in the `scope` parameter in the request body and the `Authorization` header in the request. The `Authorization` header contains your app client ID and client secret. You don't need to implement additional logic in your app to implement this solution. You must only update your configuration to change the path to your user pool token endpoint.

You can also implement token caching with ElastiCache (Redis OSS). For fine-grained control with AWS Identity and Access Management (IAM) policies, consider an Amazon DynamoDB cache.

> **Note**
>
> Caching in API Gateway is subject to additional cost. See pricing for more details.

To set up a caching proxy with API Gateway

1. Open the API Gateway console and create a REST API.
2. In **Resources**, create a POST method.
   a. Choose the HTTP **Integration type**.
   b. Select **Use HTTP proxy integration**.
   c. Enter an **Endpoint URL** of `https://<your user pool domain>/oauth2/token`.
3. In **Resources**, configure the cache key.
   a. Edit the **Method request** of your POST method.

   > **Note**
   >
   > This method request validation is for use with `client_secret_basic` authorization in token requests, where the client secret is encoded in the `Authorization` request header. For validation of the JSON request body in `client_secret_post` authorization, create instead a data model that requires that client_secret be present. In this model, your **Request validator** should **Validate body, query string parameters, and headers**.

   b. Configure the method **Request validator** to **Validate query string parameters and headers**. For more information about request validation, see Request validation in the *Amazon API Gateway Developer Guide*.
   c. Set your `scope` parameter and `Authorization` header as your caching key.
      i. Add a query string to **URL query string parameters**. Enter a query string **Name** of `scope` and select **Required** and **Caching**.
      ii. Add a header to **HTTP request headers**. Enter a request header **Name** of `Authorization` and select **Required** and **Caching**.
4. In **Stages**, configure caching.
   a. Choose the stage that you want to modify and choose **Edit** from **Stage Details**.
   b. Under **Additional settings**, **Cache settings**, turn on the **Provision API cache** option.
   c. Choose a **Cache capacity**. Higher cache capacity improves performance but comes at additional cost.
   d. Clear the **Require authorization** check box. Select **Continue**.

e. API Gateway only applies cache policies to GET methods from the stage level. You must apply a cache policy override to your POST method.

   Expand the stage you configured and select the POST method. To create cache settings for the method, choose **Create override**.

f. Activate the **Enable method cache** option.

g. Enter a **Cache time-to-live (TTL)** of 3600 seconds. Choose **Save**.

5. In **Stages**, note the **Invoke URL**.
6. Update your app to POST token requests to the **Invoke URL** of your API instead of the `/oauth2/token` endpoint of your user pool.

Scopes, M2M, and resource servers

After you configure a domain for your user pool, Amazon Cognito automatically provisions an OAuth 2.0 authorization server and a hosted web UI with sign-up and sign-in pages that your app can present to your users. For more information see User pool managed login. You can choose the scopes that you want the authorization server to add to access tokens. Scopes authorize access to resource servers and user data.

A *resource server* is an OAuth 2.0 API server. To secure access-protected resources, it validates that access tokens from your user pool contain the scopes that authorize the requested method and path in the API that it protects. It verifies the issuer based on the token signature, validity based on token expiration time, and access level based on the scopes in token claims. User pool scopes are in the access token `scope` claim. For more information about the claims in Amazon Cognito access tokens, see Understanding the access token.

With Amazon Cognito, the scopes in access tokens can authorize access to external APIs or to user attributes. You can issue access tokens to local users, federated users, or machine identities.

Topics

- API authorization
- Machine-to-machine (M2M) authorization
- About scopes
- About resource servers
- Resource binding

API authorization

The following are some of the ways that you can authorize requests to APIs with Amazon Cognito tokens:

Access token

When add an Amazon Cognito authorizer to a REST API method request configuration, add **Authorization scopes** to the authorizer configuration. With this configuration, your API accepts access tokens in the `Authorization` header and reviews them for accepted scopes.

ID token

When you pass a valid ID token to an Amazon Cognito authorizer in your REST API, API Gateway accepts the request and passes the ID token contents to the API backend.

Amazon Verified Permissions

In Verified Permissions, you have the option to create an API-linked policy store. Verified Permissions creates and assigns a Lambda authorizer that processes ID or access tokens from your request `Authorization` header. This Lambda authorizer passes your token to your policy store, where Verified Permissions compares it to policies and returns an allow or deny decision to the authorizer.

More resources

- Controlling and managing access to a REST API in API Gateway
- Authorization with Amazon Verified Permissions

Machine-to-machine (M2M) authorization

Amazon Cognito supports applications that access API data with *machine identities*. Machine identities in user pools are confidential clients that run on application servers and connect to remote APIs. Their operation happens without user interaction: scheduled tasks, data streams, or asset updates. When these clients authorize their requests with an access token, they perform *machine to machine*, or M2M, authorization. In M2M authorization, a shared secret replaces user credentials in access control.

An application that accesses an API with M2M authorization must have a client ID and client secret. In your user pool, build an app client that has a client secret. Your machine identity can

then obtain an M2M access token in one of two ways: it can request a *client credentials grant* from the Token endpoint, or it can call the GetClientToken API operation. Both approaches issue an access token that authorizes only custom scopes from resource servers. They differ in setup: the token endpoint requires a user pool domain and suits applications that use an OIDC library, while `GetClientToken` requires no domain and works through the AWS SDK, AWS CLI, or API. For more information about setting up app clients, see Application-specific settings with app clients.

The M2M access token is a verifiable statement of the operations that you want to permit your machine identity to request from an API. To learn more about how access tokens authorize API requests, continue reading. For an example application, see Amazon Cognito and API Gateway based machine to machine authorization using AWS CDK.

M2M authorization has a billing model that differs from the way that monthly active users (MAUs) are billed. Where user authentication carries a cost per active user, M2M billing reflects active M2M app clients, whether they use client credentials grants or the `GetClientToken` API operation, and total token-request volume. For more information, see Amazon Cognito Pricing. To control costs for M2M authorization, optimize the duration of access tokens and the number of token requests that your applications make. See Managing user pool token expiration and caching for a way to use API Gateway caching to reduce requests for new tokens in M2M authorization.

For information about optimizing Amazon Cognito operations that add costs to your AWS bill, see Managing costs.

Client metadata for machine-to-machine (M2M) client credentials

You can pass client metadata in M2M requests. Client metadata is additional information from a user or application environment that can contribute to the outcomes of a Pre token generation Lambda trigger. In authentication operations with a user principal, you can pass client metadata to the pre token generation trigger in the body of AdminRespondToAuthChallenge and RespondToAuthChallenge API requests. Because applications conduct the flow for generation of access tokens for M2M with direct requests to the Token endpoint, they have a different model. In the POST body of token requests for client credentials, pass an `aws_client_metadata` parameter with the client metadata object URL-encoded (`x-www-form-urlencoded`) to string. For an example request, see Client credentials with basic authorization. The following is an example parameter that passes the key-value pairs `{"environment": "dev", "language": "en-US"}`.

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

Obtaining M2M access tokens with GetClientToken

The GetClientToken API operation issues an M2M access token for a confidential app client through the AWS SDK, AWS CLI, or API, without a user pool domain. It provides the same functionality as the client credentials grant from the token endpoint; both authorize an *application* rather than a *user*.

> **Note**
>
> Amazon Cognito doesn't use AWS Identity and Access Management (IAM) identity-based policies to authorize `GetClientToken`. You can't use IAM credentials to authorize the request; authorization comes from the app client's client secret.

To use `GetClientToken`, configure an app client that has a client secret and only the `ALLOW_CLIENT_TOKEN_AUTH` authentication flow. This flow is mutually exclusive with user authentication flows. An app client can have up to two active secrets. Associate your resource server custom scopes with the app client. The following AWS CLI example creates a compatible app client.

```
aws cognito-idp create-user-pool-client \
    --user-pool-id us-west-2_EXAMPLE \
    --client-name my-m2m-client \
    --generate-secret \
    --explicit-auth-flows ALLOW_CLIENT_TOKEN_AUTH \
    --allowed-o-auth-scopes "solar-system-data/asteroids.add"
```

To get an access token, call `GetClientToken` with the app client ID, an active client secret, and the custom scopes that you want in the token. `GetClientToken` doesn't take a `UserPoolId` parameter; Amazon Cognito identifies the user pool from the client ID. If you don't specify `Scopes`, Amazon Cognito authorizes the scopes that are configured for the app client.

```
aws cognito-idp get-client-token \
    --client-id 1example23456789 \
    --secret exampleClientSecret123EXAMPLE \
    --scopes "solar-system-data/asteroids.add"
```

The response contains the access token and its metadata in a `ClientAuthenticationResult` object.

```
{
    "ClientAuthenticationResult": {
        "AccessToken": "eyJra456defEXAMPLE",
        "ExpiresIn": 3600,
        "TokenType": "Bearer"
    }
}
```

Present the access token to your resource server, for example, in the `Authorization` header of a request to an Amazon API Gateway REST API. The resource server verifies the token signature and expiration, then authorizes the request based on the scopes in the token.

To pass data to a Pre token generation Lambda trigger that customizes the token, include a `ClientMetadata` map in your request with the `--client-metadata` parameter. Your pre token generation trigger must be configured for trigger event version 3 or later to receive client metadata in the M2M flow. Amazon Cognito doesn't store, validate, or encrypt this data, and makes it available only to Lambda triggers, so don't include sensitive information.

About scopes

A *scope* is a level of access that an app can request to a resource. In an Amazon Cognito access token, the scope is backed up by the trust that you set up with your user pool: a trusted issuer of access tokens with a known digital signature. User pools can generate access tokens with scopes that prove your customer is allowed to manage some or all of their own user profile, or to retrieve data from a back-end API. Amazon Cognito user pools issue access tokens with *the user pools reserved API scope*, *custom scopes*, and *OpenID Connect (OIDC) scopes*.

The user pools reserved API scope

The `aws.cognito.signin.user.admin` scope authorizes self-service operations for the current user in the Amazon Cognito user pools API. It authorizes the bearer of an access token to query and update all information about the bearer with, for example, the GetUser and UpdateUserAttributes API operations. When you authenticate your user with the Amazon Cognito user pools API, this is the only scope you receive in the access token. It's also the only scope you need to read and write user attributes that you've authorized your app client to read and write. You can also request this scope in requests to your Authorize endpoint. This scope alone isn't sufficient to request user attributes from the userInfo endpoint. For access tokens that authorize both user pools API *and* `userInfo` requests for your users, you must request both of the scopes `openid` and `aws.cognito.signin.user.admin` in an `/oauth2/authorize` request.

Custom scopes

Custom scopes authorize requests to the external APIs that resource servers protect. You can request custom scopes with other types of scopes. You can find more information about custom scopes throughout this page.

OpenID Connect (OIDC) scopes

When you authenticate users with your user pool authorization server, including with managed login, you must request scopes. You can authenticate user pool local users and third-party federated users in your Amazon Cognito authorization server. OIDC scopes authorize your app to read user information from the userInfo endpoint of your user pool. The OAuth model, where you query user attributes from the `userInfo` endpoint, can optimize your app for a high volume of requests for user attributes. The `userInfo` endpoint returns attributes at a permission level that's determined by the scopes in the access token. You can authorize your app client to issue access tokens with the following OIDC scopes.

openid

The minimum scope for OpenID Connect (OIDC) queries. Authorizes the ID token, the unique-identifier claim `sub`, and the ability to request other scopes.

> **Note**
>
> When you request the `openid` scope and no others, your user pool ID token and `userInfo` response include claims for all user attributes that your app client can read. When you request `openid` and other OIDC scopes like `profile`, `email`, and phone, the contents of the ID token and userInfo response are limited to the constraints of the additional scopes.
> For example, a request to the Authorize endpoint with the parameter `scope=openid+email` returns an ID token with `sub`, `email`, and `email_verified`. The access token from this request returns the same attributes from userInfo endpoint. A request with parameter `scope=openid` returns all client-readable attributes in the ID token and from `userInfo`.

profile

Authorizes all user attributes that the app client can read.

email

Authorizes the user attributes `email` and `email_verified`. Amazon Cognito returns `email_verified` if it has had a value explicitly set.

phone

Authorizes the user attributes `phone_number` and `phone_number_verified`.

About resource servers

A resource server API might grant access to the information in a database, or control your IT resources. An Amazon Cognito access token can authorize access to APIs that support OAuth 2.0. Amazon API Gateway REST APIs have built-in support for authorization with Amazon Cognito access tokens. Your app passes the access token in the API call to the resource server. The resource server inspects the access token to determine if access should be granted.

Amazon Cognito might make future updates to the schema of user pool access tokens. If your app analyzes the contents of the access token before it passes it to an API, you must engineer your code to accept updates to the schema.

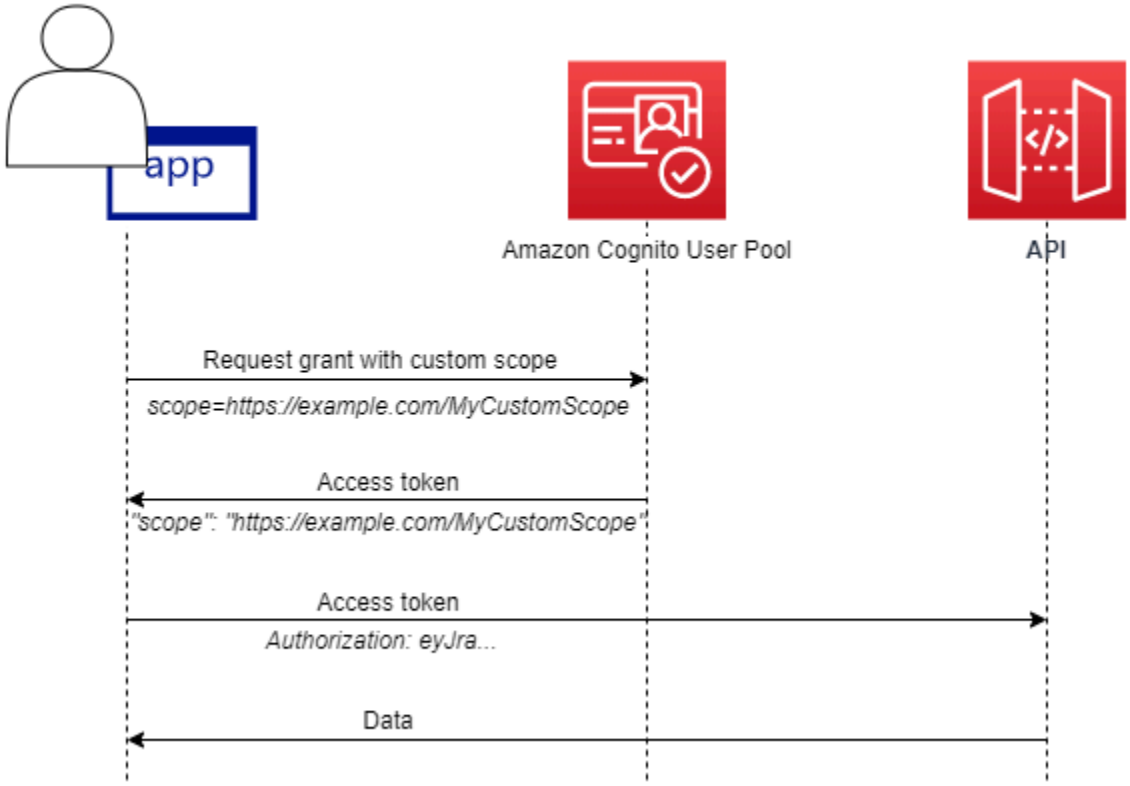
Custom scopes are defined by you, and extend the authorization capabilities of a user pool to include purposes unrelated to querying and modifying users and their attributes. For example, if you have a resource server for photos, it might define two scopes: `photos.read` for read access to the photos and `photos.write` for write/delete access. You can configure an API to accept access tokens for authorization, and grant `HTTP GET` requests to access tokens with `photos.read` in the `scope` claim, and `HTTP POST` requests to tokens with `photos.write`. These are *custom scopes*.

Note

Your resource server must verify the access token signature and expiration date before processing any claims inside the token. For more information about verifying tokens, see Verifying JSON web tokens. For more information about verifying and using user pool tokens in Amazon API Gateway, see the blog Integrating Amazon Cognito User Pools with API Gateway. API Gateway is a good option for inspecting access tokens and protecting your resources. For more about API Gateway Lambda authorizers, see Use API Gateway Lambda authorizers.

Overview

With Amazon Cognito, you can create OAuth 2.0 **Resource servers** and associate **Custom scopes** with them. Custom scopes in an access token authorize specific actions in your API. You can authorize any app client in your user pool to issue custom scopes from any of your resource servers. Associate your custom scopes with an app client and request those scopes in OAuth 2.0 authorization code grants, implicit grants, and client credentials grants from the Token endpoint. For M2M, you can also request custom scopes with the GetClientToken API operation. Amazon Cognito adds custom scopes to the `scope` claim in an access token. A client can use the access token against its resource server, which makes the authorization decision based on the scopes present in the token. For more information about access token scope, see Using Tokens with User Pools.



To get an access token with custom scopes, your app can make a request to the Token endpoint to redeem an authorization code or to request a client credentials grant. In managed login, you can also request custom scopes in an access token from an implicit grant. For M2M authorization, your app can also request custom scopes with the GetClientToken API operation.

> **Note**
>
> Because they are designed for human-interactive authentication with the user pool as the IdP, InitiateAuth and AdminInitiateAuth requests only produce a `scope` claim in the access token with the single value `aws.cognito.signin.user.admin`.

Managing the Resource Server and Custom Scopes

When creating a resource server, you must provide a resource server name and a resource server identifier. For each scope you create in the resource server, you must provide the scope name and description.

- **Resource server name**: A friendly name for the resource server, such as `Solar system object tracker` or `Photo API`.
- **Resource server identifier**: A unique identifier for the resource server. The identifier is any name that you want to associate with your API, for example `solar-system-data`. You can configure longer identifiers like `https://solar-system-data-api.example.com` as a more direct reference to API URI paths, but longer strings increase the size of access tokens.
- **Scope name**: The value that you want in your `scope` claims. For example, `sunproximity.read`.
- **Description**: A friendly description of the scope. For example, `Check current proximity to sun`.

Amazon Cognito can include custom scopes in access tokens for any users, whether they are local to your user pool or federated with a third-party identity provider. You can choose scopes for your users' access tokens during authentication flows with the OAuth 2.0 authorization server that includes managed login. Your user's authentication must begin at the Authorize endpoint with `scope` as one of the request parameters. The following is a recommended format for resource servers. For an identifier, use an API friendly name. For a custom scope, use the action that they authorize.

```
resourceServerIdentifier/scopeName
```

For example, you've discovered a new asteroid in the Kuiper belt and you want to register it through your `solar-system-data` API. The scope that authorizes write operations to the database of asteroids is `asteroids.add`. When you request the access token that will authorize

you to register your discovery, format your `scope` HTTPS request parameter as `scope=solar-system-data/asteroids.add`.

Deleting a scope from a resource server does not delete its association with all clients. Instead, the scope is marked *inactive*. Amazon Cognito doesn't add inactive scopes to access tokens, but otherwise proceeds as normal if your app requests one. If you add the scope to your resource server again later, then Amazon Cognito again writes it to the access token. If you request a scope that you haven't associated with your app client, regardless of whether you deleted it from your user pool resource server, authentication fails.

You can use the AWS Management Console, API, or CLI to define resource servers and scopes for your user pool.

Defining a resource server for your user pool (AWS Management Console)

You can use the AWS Management Console to define a resource server for your user pool.

To define a resource server

1. Sign in to the Amazon Cognito console.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **Domain** menu under **Branding** and locate **Resource servers**.
4. Choose **Create a resource server**.
5. Enter a **Resource server name**. For example, `Photo Server`.
6. Enter a **Resource server identifier**. For example, `com.example.photos`.
7. Enter **Custom scopes** for your resources, such as `read` and `write`.
8. For each **Scope name**, enter a **Description**, such as `view your photos` and `update your photos`.
9. Choose **Create**.

Your custom scopes can be reviewed in the **Domain** menu under **Resource servers**, in the **Custom scopes** column. Custom scopes can be enabled for app clients from the **App clients** menu under **Applications**. Select an app client, locate **Login pages** and choose **Edit**. Add **Custom scopes** and choose **Save changes**.

Defining a resource server for your user pool (AWS CLI and AWS API)

Use the following commands to specify resource server settings for your user pool.

To create a resource server

- AWS CLI: `aws cognito-idp create-resource-server`
- AWS API: CreateResourceServer

To get information about your resource server settings

- AWS CLI: `aws cognito-idp describe-resource-server`
- AWS API: DescribeResourceServer

To list information about all resource servers for your user pool

- AWS CLI: `aws cognito-idp list-resource-servers`
- AWS API: ListResourceServers

To delete a resource server

- AWS CLI: `aws cognito-idp delete-resource-server`
- AWS API: DeleteResourceServer

To update the settings for a resource server

- AWS CLI: `aws cognito-idp update-resource-server`
- AWS API: UpdateResourceServer

Resource binding

With resource binding, also referred to as resource indicators, you can request API-specific grants from your user pool authorization server. Resource binding is an OAuth 2.0 extension defined in RFC 8707 that allows clients to explicitly specify which resource server they intend to access during authorization requests. With resource binding, your API configurations can decline access for tokens that aren't specifically intended for them.

> **Note**
>
> You can only bind access tokens to resources for users. You can't request a resource binding with client-credentials M2M grants.

When you use resource binding with Amazon Cognito user pools, clients can include a `resource` parameter in their authentication requests to your user pool authorization server . Your user pool validates that the value of the requested resource is a URL, following the same scheme rules as app client callback URLs: `https://`, `http://` with `localhost` only, or a custom scheme like `myapp://`. Amazon Cognito sets the requested URI as the audience in the `aud` claim of the access token. If the requested resource is a user pool resource server, the resource server identifier must be in a URL format. You can request one resource per authentication request.

This feature is exclusive to managed login authentication with your user pool OAuth 2.0 authorization server. You can request resource binding in implicit and authorization-code grants from the Authorize endpoint. Token-refresh grants from the Token endpoint carry over the `aud` claim from the original request. It is not currently available in SDK authentication models.

Implement resource binding with your Amazon Cognito user pool

1. Configure one or more resource servers in your user pool with unique identifiers.
2. In your authorization request to `/oauth2/authorize`, request an authorization code or implicit grant and include the `resource` parameter. The value of `resource` must be a URL-formatted resource server identifier or a URL. For example, `&resource=https://solar-system-data-api.example.com`.
3. The authorization server validates the resource request, completes authentication, and sets the access token `aud` claim to the requested resource URL.
4. To validate that tokens were issued specifically for it, the resource that consumes your user's access token checks the `aud` claim.

Managing users

After users can sign in, you manage their accounts over the lifetime of your application. This section covers the day-to-day administration of a user directory: creating and importing users, organizing them into groups, and connecting a user pool to an identity pool so that your users can also obtain AWS credentials.

Topics

- Managing users in your user pool
- Updating user pool and app client configuration

Managing users in your user pool

After you create a user pool, you can create, confirm, and manage user accounts. With Amazon Cognito user pools groups you can manage your users and their access to resources by mapping IAM roles to groups.

Managing users in your Amazon Cognito user pool involves a variety of configuration options and administrative tasks. User pools can scale to millions of users. A user directory of this scale requires equally scalable and repeatable administrative tools. You might want to create many user profiles, manage inactive users, produce governance and compliance reports, or set up self-service tools where users do most of the work. After you create a user pool, you can control how users sign up and confirm their accounts, including requiring email or phone number verification. Administrators can also create user accounts directly and customize the welcome messages and password requirements.

User pools have user groups, where you can manage access to resources based on a user's group membership. You can assign IAM roles to these groups to manage access to AWS services with identity pools. Users' group membership is present in both ID and access tokens. With this information, you can make access-control decisions at runtime in your application or with a policy engine like Amazon Verified Permissions.

User pools often have many users. You will frequently find yourself searching for and updating user accounts. The Amazon Cognito console and API support querying users based on standard attributes like username, email, and phone number. Administrators can also reset passwords, disable accounts, and view user event history.

For migrating existing user data, Amazon Cognito has options to import users from a CSV file and to use a Lambda trigger to automatically migrate users when they first sign in. These options support user transitions from other user directories to your user pool.

You can use the user-management features in user pools to have fine-grained control over the user lifecycle and authentication experience. The combination of self-service sign-up, admin-created accounts, groups, and migration tools makes Amazon Cognito user pools a flexible user directory.

Topics

- Configuring policies for user creation
- Signing up and confirming user accounts
- Creating user accounts as administrator
- Adding groups to a user pool
- Managing and searching for user accounts
- Passwords, account recovery, and password policies
- Importing users into a user pool
- Working with user attributes

Configuring policies for user creation

Your user pool can allow users to sign up, or you can create them as an administrator. You can also control how much of the process of verification and confirmation after sign-up is in the hands of your users. For example, you might want to review sign-ups and accept them based on an external validation process. This configuration, or *admin create user policy*, also sets the amount of time before a user can no longer confirm their user account.

Amazon Cognito can serve the needs of your public customers as the customer identity and access management (CIAM) platform for your software. A user pool that accepts sign-up and has an app client, with or without managed login, creates a user profile for anyone on the internet who knows your publicly-discoverable app client ID and requests to sign up. A signed-up user profile can receive access and identity tokens and can access resources that you've authorized for your app. Before you activate sign-up in your user pool, review your options and ensure that your configuration complies with your security standards. Set **Enable self-registration** and `AllowAdminCreateUserOnly`, described in the following procedures, with care.

AWS Management Console

The **Sign-up** menu of your user pool contains some of the settings for sign-up and administrative creation of users in your user pool.

To configure the sign-up experience

1. In **Cognito-assisted verification and confirmation**, choose whether you want to **Allow Cognito to automatically send messages to verify and confirm**. With this setting enabled, Amazon Cognito sends an email or SMS message to new users with a code that they must present to your user pool. This confirms their ownership of the email address or phone number, setting the equivalent attribute as verified and confirming the user account for sign-in. The **Attributes to verify** that you choose determine the delivery methods and destinations of the verification messages.
2. **Verifying attribute changes** isn't significant when you're creating users, but relates to attribute verification. You can permit users who have changed but not yet verified their sign-in attributes to continue to sign in either with their new attribute value or with their original. For more information, see Verifying when users change their email or phone number.
3. **Required attributes** displays the attributes that must be provided a value before a user can sign up or you can create a user. You can only set required attributes when you create a user pool.
4. **Custom attributes** are important to the user creation and sign-up process because you can only set a value for *immutable* custom attributes when you first create a user. For more information about custom attributes, see Custom attributes.
5. In **Self-service sign-up**, select **Enable self-registration** if you want users to be able to generate a new account with the unauthenticated `SignUp` API. If you disable self-registration, you can only create new users as an administrator, in the Amazon Cognito console or with AdminCreateUser API requests. In a user pool where self-registration is inactive, SignUp API requests return `NotAuthorizedException` and managed login doesn't display a **Sign up** link.

For user pools where you plan to create users as an administrator, you can configure the duration of their temporary passwords in the setting in the **Authentication methods** menu under **Temporary passwords set by administrators expire in**.

Another important element of the creation of users as an administrator is the invitation message. When you create a new user, Amazon Cognito sends them a message with a link to your app so that they can sign in for the first time. Customize this message template in the **Authentication methods** menu under **Message templates**.

You can configure confidential app clients, typically web applications, with a client secret that prevents sign-up without the app client secret. As a security best practice, do not distribute app client secrets in public app clients, typically mobile apps. You can create app clients with client secrets in the **App clients** menu of the Amazon Cognito console.

Amazon Cognito user pools API

You can programmatically set the parameters for creation of users in a user pool in a CreateUserPool or UpdateUserPool API request.

The AdminCreateUserConfig element sets values for the following properties of a user pool.

1. Enable self-service sign-up
2. The invitation message that you send to new admin-created users

The following example, when added to a full API request body, sets a user pool with self-service sign-up inactive and a basic invitation email.

```
"AdminCreateUserConfig": {
      "AllowAdminCreateUserOnly": true,
      "InviteMessageTemplate": {
         "EmailMessage": "Your username is {username} and temporary password is
 {####}.",
         "EmailSubject": "Welcome to ExampleApp",
         "SMSMessage": "Your username is {username} and temporary password is
 {####}."
      }
   }
```

The following additional parameters of a CreateUserPool or UpdateUserPool API request govern the creation of new users.

AutoVerifiedAttributes

The attributes, email addresses or phone numbers, that you want to automatically send a message to when you register a new user.

Policies

The user pool password policy.

Schema

The user pool custom attributes. They are important to the user creation and sign-up process because you can only set a value for *immutable* custom attributes when you first create a user.

This parameter also sets the required attributes for your user pool. The following text, when inserted into the `Schema` element of a full API request body, set the `email` attribute as required.

```
{
            "Name": "email",
            "Required": true
}
```

Signing up and confirming user accounts

User accounts are added to your user pool in one of the following ways:

- The user signs up in your user pool's client app. This can be a mobile or web app.
- You can import the user's account into your user pool. For more information, see Importing users into user pools from a CSV file.
- You can create the user's account in your user pool and invite the user to sign in. For more information, see Creating user accounts as administrator.

Users who sign themselves up must be confirmed before they can sign in. Imported and created users are already confirmed, but they must create their password the first time they sign in. The following sections explain the confirmation process and email and phone verification.

Passwords at sign-up

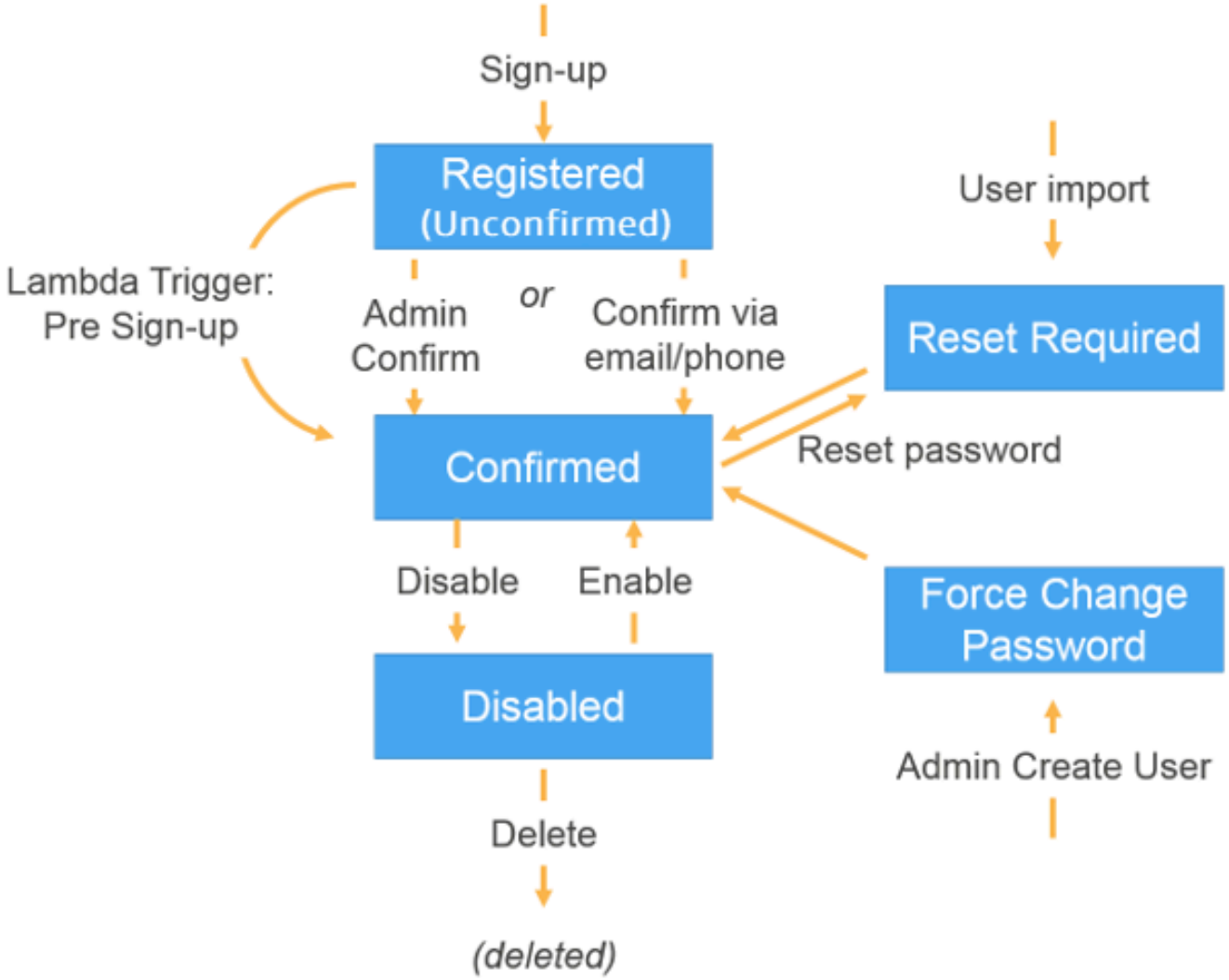
Amazon Cognito requires passwords from all users when they sign up, except under the following conditions. If *all* of these conditions are met, you can omit passwords in sign-up operations.

1. Passwordless sign-in is active in your user pool and app client.

2. Your application is custom-built with authentication modules in an AWS SDK. Managed login and the hosted UI always require passwords.
3. Users provide attribute values for the passwordless sign-in methods—email or SMS message one-time passwords (OTPs)—that you permit. For example, if you allow sign-in with email and phone OTP, users can provide either a phone number or email address, but if you only allow sign-in with email, they must provide an email address.
4. Your user pool automatically verifies the attributes that users can use with passwordless sign-in.
5. For any given SignUp request, the user doesn't provide a value for the Password parameter.

Overview of user account confirmation

The following diagram illustrates the confirmation process:



A user account can be in any of the following states:

Registered (Unconfirmed)

The user has successfully signed up, but cannot sign in until the user account is confirmed. The user is enabled but not confirmed in this state.

New users who sign themselves up start in this state.

Confirmed

The user account is confirmed and the user can sign in. When a user enters a code or follows an email link to confirm their user account, that email or phone number is automatically verified. The code or link is valid for 24 hours.

If the user account was confirmed by the administrator or a pre sign-up Lambda trigger, there might not be a verified email or phone number associated with the account.

Password Reset Required

The user account is confirmed, but the user must request a code and reset their password before they can sign in.

User accounts that are imported by an administrator or developer start in this state.

Force Change Password

The user account is confirmed and the user can sign in using a temporary password, but on first sign-in, the user must change their password to a new value before doing anything else.

User accounts that are created by an administrator or developer start in this state.

Disabled

Before you can delete a user account, you must disable sign-in access for that user.

More resources

- Detecting and remediating inactive user accounts with Amazon Cognito

Verifying contact information at sign-up

When new users sign up in your app, you probably want them to provide at least one contact method. For example, with your users' contact information, you might:

- Send a temporary password when a user chooses to reset their password.
- Notify users when their personal or financial information is updated.
- Send promotional messages, such as special offers or discounts.
- Send account summaries or billing reminders.

For use cases like these, it's important that you send your messages to a verified destination. Otherwise, you might send your messages to an invalid email address or phone number that was typed incorrectly. Or worse, you might send sensitive information to bad actors who pose as your users.

To help ensure that you send messages only to the right individuals, configure your Amazon Cognito user pool so that users must provide the following when they sign up:

a. An email address or phone number.
b. A verification code that Amazon Cognito sends to that email address or phone number. If 24 hours have passed and your user's code or link is no longer valid, call the ResendConfirmationCode API operation to generate and send a new code or link.

By providing the verification code, a user proves that they have access to the mailbox or phone that received the code. After the user provides the code, Amazon Cognito updates the information about the user in your user pool by:

- Setting the user's status to `CONFIRMED`.
- Updating the user's attributes to indicate that the email address or phone number is verified.

To view this information, you can use the Amazon Cognito console. Or, you can use the `AdminGetUser` API operation, the `admin-get-user` command with the AWS CLI, or a corresponding action in one of the AWS SDKs.

If a user has a verified contact method, Amazon Cognito automatically sends a message to the user when the user requests a password reset.

Other actions that confirm and verify user attributes

The following user activity verifies user attributes. You're not required to set these attributes to automatically verify: the listed actions mark them as verified in all cases.

Email address

1. Successfully completing passwordless authentication with an email one-time password (OTP).
2. Successfully completing multi-factor authentication (MFA) with an email OTP.

Phone number

1. Successfully completing passwordless authentication with an SMS OTP.
2. Successfully completing MFA with an SMS OTP.

To configure your user pool to require email or phone verification

When you verify your users' email addresses and phone numbers, you ensure that you can contact your users. Complete the following steps in the AWS Management Console to configure your user pool to require that your users confirm their email addresses or phone numbers.

> **Note**
>
> If you don't yet have a user pool in your account, see Create a new application in the Amazon Cognito console.

To configure your user pool

1. Navigate to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. From the navigation pane, choose **User Pools**. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Sign-up** menu and locate **Attribute verification and user account confirmation**. Choose **Edit**.
4. Under **Cognito-assisted verification and confirmation**, choose whether you will **Allow Cognito to automatically send messages to verify and confirm**. With this setting enabled, Amazon Cognito sends messages to the user contact attributes you choose when a user signs up, or you create a user profile. To verify attributes and confirm user profiles for sign-in, Amazon Cognito sends a code or link in messages to users. The users must then enter the code in your UI so that your app can confirm them in a `ConfirmSignUp` or `AdminConfirmSignUp` API request.

> **Note**
>
> You can also disable **Cognito-assisted verification and confirmation** and use authenticated API actions or Lambda triggers to verify attributes and confirm users. If you choose this option, Amazon Cognito doesn't send verification codes when users sign up. Choose this option if you are using a custom authentication flow that verifies at least one contact method without using verification codes from Amazon Cognito. For example, you might use a pre sign-up Lambda trigger that automatically verifies email addresses that belong to a specific domain.
> If you don't verify your users' contact information, they may be unable to use your app. Remember that users require verified contact information to:
>
> - **Reset their passwords** — When a user chooses an option in your app that calls the `ForgotPassword` API action, Amazon Cognito sends a temporary password to the user's email address or phone number. Amazon Cognito sends this password only if the user has at least one verified contact method.
> - **Sign in by using an email address or phone number as an alias** — If you configure your user pool to allow these aliases, then a user can sign in with an alias only if the alias is verified. For more information, see Customizing sign-in attributes.

5. Choose your **Attributes to verify**:

Send SMS message, verify phone number

Amazon Cognito sends a verification code in an SMS message when the user signs up. Choose this option if you typically communicate with your users through SMS messages. For example, you will want to use verified phone numbers if you send delivery notifications, appointment confirmations, or alerts. User phone numbers will be the verified attribute when accounts are confirmed; you must take additional action to verify and communicate with user email addresses.

Send email message, verify email address

Amazon Cognito sends a verification code through an email message when the user signs up. Choose this option if you typically communicate with your users through email. For example, you will want to use verified email addresses if you send billing statements, order summaries, or special offers. User email addresses will be the verified attribute when

accounts are confirmed; you must take additional action to verify and communicate with user phone numbers.

Send SMS message if phone number is available, otherwise send email message

Choose this option if you don't require all users to have the same verified contact method. In this case, the sign-up page in your app could ask users to verify only their preferred contact method. When Amazon Cognito sends a verification code, it sends the code to the contact method provided in the `SignUp` request from your app. If a user provides both an email address and a phone number, and your app provides both contact methods in the `SignUp` request, Amazon Cognito sends a verification code only to the phone number.

If you require users to verify both an email address and a phone number, choose this option. Amazon Cognito verifies one contact method when the user signs up, and your app must verify the other contact method after the user signs in. For more information, see If you require users to confirm both email addresses and phone numbers.

6. Choose **Save changes**.

Authentication flow with email or phone verification

If your user pool requires users to verify their contact information, your app must facilitate the following flow when a user signs up:

1. A user signs up in your app by entering a username, phone number and/or email address, and possibly other attributes.
2. The Amazon Cognito service receives the sign-up request from the app. After verifying that the request contains all attributes required for sign-up, the service completes the sign-up process and sends a confirmation code to the user's phone (in an SMS message) or email. The code is valid for 24 hours.
3. The service returns to the app that sign-up is complete and that the user account is pending confirmation. The response contains information about where the confirmation code was sent. At this point the user's account is in an unconfirmed state, and the user's email address and phone number are unverified.
4. The app can now prompt the user to enter the confirmation code. It is not necessary for the user to enter the code immediately. However, the user will not be able to sign in until after they enter the confirmation code.
5. The user enters the confirmation code in the app.

6. The app calls ConfirmSignUp to send the code to the Amazon Cognito service, which verifies the code and, if the code is correct, sets the user's account to the confirmed state. After successfully confirming the user account, the Amazon Cognito service automatically marks the attribute that was used to confirm (email address or phone number) as verified. Unless the value of this attribute is changed, the user will not have to verify it again.
7. At this point the user's account is in a confirmed state, and the user can sign in.

If you require users to confirm both email addresses and phone numbers

Amazon Cognito verifies only one contact method when a user signs up. In cases where Amazon Cognito must choose between verifying an email address or phone number, it chooses to verify the phone number by sending a verification code through SMS message. For example, if you configure your user pool to allow users to verify either email addresses or phone numbers, and if your app provides both of these attributes upon sign-up, Amazon Cognito verifies only the phone number. After a user verifies their phone number, Amazon Cognito sets the user's status to `CONFIRMED`, and the user is allowed to sign in to your app.

After the user signs in, your app can provide the option to verify the contact method that wasn't verified during sign-up. To verify this second method, your app calls the `VerifyUserAttribute` API action. Note that this action requires an `AccessToken` parameter, and Amazon Cognito only provides access tokens for authenticated users. Therefore, you can verify the second contact method only after the user signs in.

If you require your users to verify both email addresses and phone numbers, do the following:

1. Configure your user pool to allow users to verify email address or phone numbers.
2. In the sign-up flow for your app, require users to provide both an email address and a phone number. Call the SignUp API action, and provide the email address and phone number for the `UserAttributes` parameter. At this point, Amazon Cognito sends a verification code to the user's phone.
3. In your app interface, present a confirmation page where the user enters the verification code. Confirm the user by calling the ConfirmSignUp API action. At this point, the user's status is `CONFIRMED`, and the user's phone number is verified, but the email address is not verified.
4. Present the sign-in page, and authenticate the user by calling the InitiateAuth API action. After the user is authenticated, Amazon Cognito returns an access token to your app.
5. Call the GetUserAttributeVerificationCode API action. Specify the following parameters in the request:

- `AccessToken` – The access token returned by Amazon Cognito when the user signed in.
- `AttributeName` – Specify `"email"` as the attribute value.

Amazon Cognito sends a verification code to the user's email address.

6. Present a confirmation page where the user enters the verification code. When the user submits the code, call the `VerifyUserAttribute` API action. Specify the following parameters in the request:

   - `AccessToken` – The access token returned by Amazon Cognito when the user signed in.
   - `AttributeName` – Specify `"email"` as the attribute value.
   - `Code` – The verification code that the user provided.

   At this point, the email address is verified.

Allowing users to sign up in your app but confirming them as a user pool administrator

You might not want your user pool to automatically send verification messages in your user pool, but still want to allow anyone to sign up for an account. This model leaves room, for example, for human review of new sign-up requests, and for batch validation and processing of sign-ups. You can confirm new user accounts in the Amazon Cognito console or with the IAM-authenticated API operation AdminConfirmSignUp. You can confirm user accounts as an administrator whether or not your user pool sends verification messages.

You can only confirm a user self-service sign-up with this technique. To confirm a user that you create as an administrator, create an AdminSetUserPassword API request with `Permanent` set to `True`.

1. A user signs up in your app by entering a username, phone number and/or email address, and possibly other attributes.
2. The Amazon Cognito service receives the sign-up request from the app. After verifying that the request contains all attributes required for sign-up, the service completes the sign-up process and returns to the app that sign-up is complete, pending confirmation. At this point the user's account is in an unconfirmed state. The user cannot sign in until the account is confirmed.

3. Confirm the user's account. You must sign in to the AWS Management Console or sign your API request with AWS credentials to confirm the account.

   a. To confirm a user in the Amazon Cognito console, navigate to the **Users** menu, choose the user who you want to confirm, and from the **Actions** menu select **Confirm**.

   b. To confirm a user in the AWS API or CLI, create a AdminConfirmSignUp API request, or admin-confirm-sign-up in the AWS CLI.

4. At this point the user's account is in a confirmed state, and the user can sign in.

Computing secret hash values

Assign a client secret to your confidential app client as a best practice. When you assign a client secret to your app client, your Amazon Cognito user pools API requests must include a hash that includes the client secret in the request body. To validate your knowledge of the client secret for the API operations in the following lists, concatenate the client secret with your app client ID and your user's username, then base64-encode that string.

When your app signs in users to a client that has a secret hash, you can use the value of any user pool sign-in attribute as the username element of the secret hash. When your app requests new tokens in an authentication operation with `REFRESH_TOKEN_AUTH`, the value of the username element depends on your sign-in attributes. When your user pool doesn't have `username` as a sign-in attribute, set the secret hash username value from the user's `sub` claim from their access or ID token. When `username` is a sign-in attribute, set the secret hash username value from the `username` claim.

The following Amazon Cognito user pools APIs accept a client-secret hash value in a `SecretHash` parameter.

- ConfirmForgotPassword
- ConfirmSignUp
- ForgotPassword
- ResendConfirmationCode
- SignUp

Additionally, the following APIs accept a client-secret hash value in a `SECRET_HASH` parameter, either in authentication parameters or in a challenge response.

API operation	Parent parameter for SECRET_HASH
InitiateAuth	AuthParameters
AdminInitiateAuth	AuthParameters
RespondToAuthChallenge	ChallengeResponses
AdminRespondToAuthChallenge	ChallengeResponses

The secret hash value is a Base 64-encoded keyed-hash message authentication code (HMAC) calculated using the secret key of a user pool client and username plus the client ID in the message. The following pseudocode shows how this value is calculated. In this pseudocode, + indicates concatenation, HMAC_SHA256 represents a function that produces an HMAC value using HmacSHA256, and Base64 represents a function that produces Base-64-encoded version of the hash output.

```
Base64 ( HMAC_SHA256 ( "Client Secret Key", "Username" + "Client Id" ) )
```

For a detailed overview of how to calculate and use the SecretHash parameter, see How do I troubleshoot "Unable to verify secret hash for client <client-id>" errors from my Amazon Cognito user pools API? in the AWS Knowledge Center.

You can use the following code examples in your server-side app code.

Shell

```
echo -n "[username][app client ID]" | openssl dgst -sha256 -hmac [app client secret]
 -binary | openssl enc -base64
```

Java

```
import javax.crypto.Mac;
import javax.crypto.spec.SecretKeySpec;

public static String calculateSecretHash(String userPoolClientId, String
 userPoolClientSecret, String userName) {
    final String HMAC_SHA256_ALGORITHM = "HmacSHA256";
```

```
    SecretKeySpec signingKey = new SecretKeySpec(
            userPoolClientSecret.getBytes(StandardCharsets.UTF_8),
            HMAC_SHA256_ALGORITHM);
    try {
        Mac mac = Mac.getInstance(HMAC_SHA256_ALGORITHM);
        mac.init(signingKey);
        mac.update(userName.getBytes(StandardCharsets.UTF_8));
        byte[] rawHmac =
 mac.doFinal(userPoolClientId.getBytes(StandardCharsets.UTF_8));
        return Base64.getEncoder().encodeToString(rawHmac);
    } catch (Exception e) {
        throw new RuntimeException("Error while calculating ");
    }
}
```

Python

```
import sys
import hmac, hashlib, base64
username = sys.argv[1]
app_client_id = sys.argv[2]
key = sys.argv[3]
message = bytes(sys.argv[1]+sys.argv[2],'utf-8')
key = bytes(sys.argv[3],'utf-8')
secret_hash = base64.b64encode(hmac.new(key, message,
 digestmod=hashlib.sha256).digest()).decode()
print("SECRET HASH:",secret_hash)
```

Confirming user accounts without verifying email or phone number

The pre sign-up Lambda trigger can be used to auto-confirm user accounts at sign-up, without requiring a confirmation code or verifying email or phone number. Users who are confirmed this way can immediately sign in without having to receive a code.

You can also mark a user's email or phone number verified through this trigger.

> **Note**
>
> While this approach is convenient for users when they're getting started, we recommend auto-verifying at least one of email or phone number. Otherwise the user can be left unable to recover if they forget their password.

If you don't require the user to receive and enter a confirmation code at sign-up and you don't auto-verify email and phone number in the pre sign-up Lambda trigger, you risk not having a verified email address or phone number for that user account. The user can verify the email address or phone number at a later time. However, if the user forgets his or her password and doesn't have a verified email address or phone number, the user is locked out of the account, because the forgot-password flow requires a verified email or phone number in order to send a verification code to the user.

Verifying when users change their email or phone number

In user pools that you configure with multiple sign-in names, users can enter a phone number or an email address as their username at sign-in. When they update their email address or phone number in your app, Amazon Cognito can immediately send them a message with a code that verifies their ownership of the new attribute value. To enable automatic sending of these verification codes, see Configuring email or phone verification.

Users who receive a verification code must provide that code back to Amazon Cognito in a VerifyUserAttribute request. After they provide the code, their attribute is marked as verified. Typically, when users update their email address or phone number, you'll want to verify that they own the new value before they can use it to sign in and receive messages. User pools have a configurable option that determines whether users must verify updates to their email address or phone number.

This option is the user pool property `AttributesRequireVerificationBeforeUpdate`. Configure it in a CreateUserPool or UpdateUserPool request, or with the setting **Keep original attribute value active when an update is pending** in the **Sign-up** menu of the Amazon Cognito console.

How your user pool treats updates to email addresses and phone numbers is connected to the username configuration of your user pool. User pool usernames can be in a *username attributes* configuration where sign-in names are email address, phone number, or both. They can also be in an *alias attributes* configuration where the `username` attribute is a sign-in name along with email

address, phone number, or preferred username as alternative sign-in names. For more information, see Customizing sign-in attributes.

You can also use a custom message Lambda trigger to customize the verification message. For more information, see Custom message Lambda trigger. When a user's email address or phone number is unverified, your application should inform the user that they must verify the attribute, and provide a button or link for users to enter their verification code.

The following table describes how `AttributesRequireVerificationBeforeUpdate` and alias settings determine the outcome when users change the value of their sign-in attributes.

Username configuration	Behavior when users must verify new attributes	Behavior when users aren't required to verify new attributes
Username attributes	Original attribute remains verified, eligible for sign-in, and at original value. When user verifies new value, Amazon Cognito updates the attribute value, marks it verified, and makes it eligible for sign-in.	Amazon Cognito updates attribute to new value. New value is eligible for sign-in. When user verifies new value, Amazon Cognito marks it as verified.
Alias attributes	Original attribute remains verified, eligible for sign-in, and at original value. When user verifies new value, Amazon Cognito updates the attribute value, marks it verified, and makes it eligible for sign-in.	Amazon Cognito updates attribute to new value. Neither original or new attribute value is eligible for sign-in. When user verifies new value, Amazon Cognito updates the attribute value, marks it verified, and makes it eligible for sign-in.

Example 1

User 1 signs into your application with the email address `user1@example.com` and has the username `user1` (alias attributes). Your user pool is configured to verify updates

to sign-in attributes and to automatically send verification messages. They request to update their email address to `user1+foo@example.com`. They receive a verification email at `user1+foo@example.com` and *can sign in again* only with the email address `user1@example.com`. Later, they enter their verification code and can sign in again only with the email address `user1+foo@example.com`.

Example 2

User 2 signs into your application with the email address `user2@example.com` and has a username (alias attributes). Your user pool is configured *not* to verify updates to sign-in attributes and to automatically send verification messages. They request to update their email address to `user2+bar@example.com`. They receive a verification email at `user2+bar@example.com` and *can't sign in again*. Later, they enter their verification code and can sign in again only with the email address `user2+bar@example.com`.

Example 3

User 3 signs into your application with the email address `user3@example.com` and doesn't have a username (username attributes). Your user pool is configured *not* to verify updates to sign-in attributes and to automatically send verification messages. They request to update their email address to `user3+baz@example.com`. They receive a verification email at `user3+baz@example.com`, but they *can immediately sign in* with no additional action taken with the verification code.

Confirmation and verification processes for user accounts created by administrators or developers

User accounts that are created by an administrator or developer are already in the confirmed state, so users aren't required to enter a confirmation code. The invitation message that the Amazon Cognito service sends to these users includes the username and a temporary password. The user is required to change the password before signing in. For more information, see the Customize email and SMS messages in Creating user accounts as administrator and the Custom Message trigger in Customizing user pool workflows with Lambda triggers.

Confirmation and verification processes for imported user accounts

User accounts that are created by using the user import feature in the AWS Management Console, CLI, or API (see Importing users into user pools from a CSV file) are already in the confirmed state,

so users aren't required to enter a confirmation code. No invitation message is sent. However, imported user accounts require users to first request a code by calling the `ForgotPassword` API and then create a password using the delivered code by calling `ConfirmForgotPassword` API before they sign in. For more information, see Requiring imported users to reset their passwords.

Either the user's email or phone number must be marked as verified when the user account is imported, so no verification is required when the user signs in.

Sending emails while testing your app

Amazon Cognito sends email messages to your users when they create and manage their accounts in the client app for your user pool. If you configure your user pool to require email verification, Amazon Cognito sends an email when:

- A user signs up.
- A user updates their email address.
- A user performs an action that calls the `ForgotPassword` API action.
- You create a user account as an administrator.

Depending on the action that initiates the email, the email contains a verification code or a temporary password. Your users must receive these emails and understand the message. Otherwise, they might be unable to sign in and use your app.

To ensure that emails send successfully and that the message looks correct, test the actions in your app that initiate email deliveries from Amazon Cognito. For example, by using the sign-up page in your app, or by using the `SignUp` API action, you can initiate an email by signing up with a test email address. When you test in this way, remember the following:

Important

When you use an email address to test actions that initiate emails from Amazon Cognito, don't use a fake email address (one that has no mailbox). Use a real email address that will receive the email from Amazon Cognito without creating a *hard bounce*.
A hard bounce occurs when Amazon Cognito fails to deliver the email to the recipient's mailbox, which always happens if the mailbox doesn't exist.
Amazon Cognito limits the number of emails that can be sent by AWS accounts that persistently incur hard bounces.

When you test actions that initiate emails, use one of the following email addresses to prevent hard bounces:

- An address for an email account that you own and use for testing. When you use your own email address, you receive the email that Amazon Cognito sends. With this email, you can use the verification code to test the sign-up experience in your app. If you customized the email message for your user pool, you can check that your customizations look correct.
- The mailbox simulator address, *success@simulator.amazonses.com*. If you use the simulator address, Amazon Cognito sends the email successfully, but you're not able to view it. This option is useful when you don't need to use the verification code and you don't need to check the email message.
- The mailbox simulator address with the addition of an arbitrary label, such as *success+user1@simulator.amazonses.com* or *success+user2@simulator.amazonses.com*. Amazon Cognito emails these addresses successfully, but you're not able to view the emails that it sends. This option is useful when you want to test the sign-up process by adding multiple test users to your user pool, and each test user has a unique email address.

Configuring email or phone verification

You can choose settings for email or phone verification under the **Authentication methods** menu. For more information on multi-factor authentication (MFA), see SMS Text Message MFA.

Amazon Cognito uses Amazon SNS to send SMS messages. If you haven't sent an SMS message from Amazon Cognito or any other AWS service before, Amazon SNS might place your account in the SMS sandbox. We recommend that you send a test message to a verified phone number before you remove your account from the sandbox to production. Additionally, if you plan to send SMS messages to US destination phone numbers, you must obtain an origination or Sender ID from Amazon Pinpoint. To configure your Amazon Cognito user pool for SMS messages, see SMS message settings for Amazon Cognito user pools.

Amazon Cognito can automatically verify email addresses or phone numbers. To do this verification, Amazon Cognito sends a verification code or a verification link. For email addresses, Amazon Cognito can send a code or a link in an email message. You can choose a **Verification type** of **Code** or **Link** when you edit your **Verification message** template in the **Message templates** menu in the Amazon Cognito console. For more information, see Customizing email verification messages.

For phone numbers, Amazon Cognito sends a code in an SMS text message.

Amazon Cognito must verify a phone number or email address to confirm users and help them to recover forgotten passwords. Alternatively, you can automatically confirm users with the pre sign-up Lambda trigger or use the AdminConfirmSignUp API operation. For more information, see Signing up and confirming user accounts.

The verification code or link is valid for 24 hours.

If you choose to require verification for an email address or phone number, Amazon Cognito automatically sends the verification code or link when a user signs up. If the user pool has a Custom SMS sender Lambda trigger or Custom email sender Lambda trigger configured, that function is invoked instead.

Notes

- Amazon SNS charges separately for SMS text messaging that it uses to verify phone numbers. There is no charge to send email messages. For information about Amazon SNS pricing, see Worldwide SMS pricing. For the current list of countries where SMS messaging is available, see Supported regions and countries.
- When you test actions in your app that generate email messages from Amazon Cognito, use a real email address that Amazon Cognito can reach without hard bounces. For more information, see the section called “Sending emails while testing your app”.
- The forgotten password flow requires either the user's email or the user's phone number to verify the user.

Important

If a user signs up with both a phone number and an email address, and your user pool settings require verification of both attributes, Amazon Cognito sends a verification code to the phone number through SMS message. Amazon Cognito hasn't yet verified the email address, so your app must call GetUser to see if an email address awaits verification. If it does require verification, the app must call GetUserAttributeVerificationCode to initiate the email verification flow. Then it must submit the verification code by calling VerifyUserAttribute.

You can adjust your SMS message spend quota for an AWS account and for individual messages. The limits apply only to the cost to send SMS messages. For more information, see **What are account-level and message-level spend quotas and how do they work?** in the Amazon SNS FAQs.

Amazon Cognito sends SMS messages using Amazon SNS resources in either the AWS Region where you created the user pool or in a **Legacy Amazon SNS alternate Region** from the following table. The exception is Amazon Cognito user pools in the Asia Pacific (Seoul) Region. These user pools use your Amazon SNS configuration in the Asia Pacific (Tokyo) Region. For more information, see Choose the AWS Region for SMS messages.

Amazon Cognito Region	Legacy Amazon SNS alternate Region
US East (Ohio)	US East (N. Virginia)
Asia Pacific (Mumbai)	Asia Pacific (Singapore)
Asia Pacific (Seoul)	Asia Pacific (Tokyo)
Canada (Central)	US East (N. Virginia)
Europe (Frankfurt)	Europe (Ireland)
Europe (London)	Europe (Ireland)

Example: If your Amazon Cognito user pool is in Asia Pacific (Mumbai), and you have increased your spend limit in ap-southeast-1, you might not want to request a separate increase in ap-south-1. Instead, you can use your Amazon SNS resources in Asia Pacific (Singapore).

Verifying updates to email addresses and phone numbers

An email address or phone number attribute can become active and unverified immediately after your user changes its value. Amazon Cognito can also require that your user verifies the new value before Amazon Cognito updates the attribute. When you require that your users first verify the new value, they can use the original value for sign-in and to receive messages until they verify the new value.

When your users can use their email address or phone number as a sign-in alias in your user pool, their sign-in name for an updated attribute depends on whether you require verification of updated attributes. When you require that users verify an updated attribute, a user can sign in with

the original attribute value until they verify the new value. When you don't require that users verify an updated attribute, a user can't sign in or receive messages at either the new or the original attribute value until they verify the new value.

For example, your user pool allows sign-in with an email address alias, and requires that users verify their email address when they update. Sue, who signs in as `sue@example.com`, wants to change her email address to `sue2@example.com` but accidentally enters **s**`sue2@example.com`. Sue doesn't receive the verification email, so she can't verify `ssue2@example.com`. Sue signs in as `sue@example.com` and resubmits the form in your app to update her email address to `sue2@example.com`. She receives this email, provides the verification code to your app, and begins signing in as `sue2@example.com`.

When a user updates an attribute and your user pool verifies new attribute values

- They can sign in with the original attribute value before they have confirmed the code to verify the new value.
- They can only sign in with the new attribute value after they have confirmed the code to verify the new value.
- If you set `email_verified` or `phone_number_verified` to `true` in an AdminUpdateUserAttributes API request, they can sign in before they have confirmed the code that Amazon Cognito sent to them.

When a user updates an attribute and your user pool doesn't verify new attribute values

- They can't sign in with, or receive messages at, the original attribute value.
- They can't sign in with, or receive messages other than a confirmation code at, the new attribute value before they have confirmed the code to verify the new value.
- If you set `email_verified` or `phone_number_verified` to `true` in an AdminUpdateUserAttributes API request, they can sign in before they have confirmed the code that Amazon Cognito sent to them.

To require attribute verification when users update their email address or phone number

1. Sign in to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.

3. In the **Sign-up** menu, choose **Edit** under **Attribute verification and user account confirmation**.
4. Choose **Keep original attribute value active when an update is pending**.
5. Under **Active attribute values when an update is pending**, choose the attributes that you want to require your users verify before Amazon Cognito updates the value.
6. Choose **Save changes**.

To require attribute update verification with the Amazon Cognito API, you can set the `AttributesRequireVerificationBeforeUpdate` parameter in an UpdateUserPool request.

Authorizing Amazon Cognito to send SMS messages on your behalf

To send SMS messages to your users on your behalf, Amazon Cognito needs your permission. To grant that permission, you can create an AWS Identity and Access Management (IAM) role. In the **Authentication methods** menu of the Amazon Cognito console under SMS, choose **Edit** to set a role.

Configuring MFA, authentication, verification and invitation messages

With Amazon Cognito, you can customize SMS and email authentication, verification, and user invitation messages to enhance the security and user experience of your application. You can choose between code-based and one-click link verifications for some messages. This topic discusses how you can personalize authentication and verification communications in the Amazon Cognito console.

In the **Message templates** menu, you can customize:

- Your email and SMS message templates for one-time password (OTP) and multi-factor (MFA) authentication
- Your SMS and email verification messages
- The verification type for email—code or link

> **Note**
>
> Amazon Cognito sends links with your link-based template in the verification messages when users sign up or resend a confirmation code. Emails from attribute-update and password-reset operations use the code template.

- Your user invitation messages
- FROM and REPLY-TO email addresses for emails going through your user pool

> **Note**
>
> The SMS and email verification message templates only appear if you have chosen to require phone number and email verification. Similarly, the SMS MFA message template only appears if the MFA setting is **required** or **optional**.

Topics

- Message templates
- Customizing email and SMS MFA messages
- Customizing email verification messages
- Customizing user invitation messages
- Customizing your email address
- Authorizing Amazon Cognito to send Amazon SES email on your behalf (from a custom FROM email address)

Message templates

You can use message templates to insert placeholders into your messages. Amazon Cognito replace the placeholders with the corresponding values. You can reference *Universal template placeholders* in message templates of any type, although these values won't be present in all message types.

Universal template placeholders

Description	Token	Message type
Verification code	{####}	Verification, confirmation, and MFA messages
Temporary password	{####}	Forgot-password and invitation messages

Description	Token	Message type
User name	`{username}`	Invitation and advanced security messages

One of the available automated responses with threat protection is to notify the user that Amazon Cognito detected potentially-malicious activity. You can use advanced security template placeholders to do the following:

- Include specific details about an event such as IP address, city, country, sign-in time, and device name. Amazon Cognito threat protection can analyze these details.
- Verify whether a one-click link is valid.
- Use event ID, feedback token, and user name to build your own one-click link.

> **Note**
>
> To generate one-click links and use the `{one-click-link-valid}` and `{one-click-link-invalid}` placeholders in advanced security email templates, you must already have a domain configured for your user pool.

Threat protection adds the following placeholders that you can insert into the message templates. These placeholders apply to **Adaptive authentication messages**, notifications that Amazon Cognito sends to users whose sessions have been evaluated for a level of risk. To configure message templates with these variables, update the **Full-function** configuration of your threat protection in the Amazon Cognito console, or submit templates in a SetRiskConfiguration request.

Advanced security template placeholders

Description	Token
IP address	`{ip-address}`
City	`{city}`
Country	`{country}`

Description	Token
Log-in time	`{login-time}`
Device name	`{device-name}`
One-click link is valid	`{one-click-link-valid}`
One-click link is not valid	`{one-click-link-invalid}`
Event ID	`{event-id}`
Feedback token	`{feedback-token}`

Customizing email and SMS MFA messages

To customize the SMS and email messages for multi-factor authentication (MFA), edit **MFA message** from the **Message templates** menu in the Amazon Cognito user pools console.

> **Important**
>
> Your custom message must contain the `{####}` placeholder. This placeholder is replaced with the authentication code before the message is sent.

Amazon Cognito sets a maximum length for SMS messages, including the authentication code, of 140 UTF-8 characters.

Customizing SMS verification messages

To customize the SMS message for phone number verification, edit the **Verification message** template from the **Message templates** menu of your user pool.

> **Important**
>
> Your custom message must contain the `{####}` placeholder. This placeholder is replaced with the verification code before the message is sent.

The maximum length for the message, including the verification code, is 140 UTF-8 characters.

Customizing email verification messages

To verify the email address of a user in your user pool with Amazon Cognito, you can send the user an email message with a link that they can select, or you can send them a code that they can enter.

To customize the email subject and message content for email address verification messages, edit the **Verification message** template in the **Message templates** menu of your user pool. You can choose a **Verification type** of **Code** or **Link** when you edit your **Verification message** template.

When you choose **Code** as the verification type, your custom message must contain the `{####}` placeholder. When you send the message, the verification code replaces this placeholder.

When you choose **Link** as the verification type, your custom message must include a placeholder in the format `{##Verify Your Email##}`. You can change the text string between the placeholder characters, for example `{##Click here##}`. A verification link titled *Verify Your Email* replaces this placeholder.

The link for an email verification message directs your user to a URL like the following example.

```
https://<your user pool domain>/confirmUser/?
client_id=abcdefg12345678&user_name=emailtest&confirmation_code=123456
```

The maximum length for the message, including the verification code (if present), is 20,000 UTF-8 characters. You can use HTML tags in this message to format the contents.

Customizing user invitation messages

You can customize the user invitation message that Amazon Cognito sends to new users by SMS or email message by editing the **Invitation messages** template in the **Message templates** menu.

> **Important**
>
> Your custom message must contain the `{username}` and `{####}` placeholders. When Amazon Cognito sends the invitation message, it replaces these placeholders with your user's user name and password.

The maximum length of an SMS message, including the verification code, is 140 UTF-8 characters. The maximum length of an email message, including the verification code, is 20,000 UTF-8 characters. You may use HTML tags in your email messages to format the contents.

Customizing your email address

By default, Amazon Cognito sends email messages to users in your user pools from the address **no-reply@verificationemail.com**. You can choose to specify custom FROM and REPLY-TO email addresses instead of **no-reply@verificationemail.com**.

To customize the FROM and REPLY-TO email addresses

1. Navigate to the Amazon Cognito console, and choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Authentication methods** menu. Under **Email**, choose **Edit**.
4. Choose an **SES Region**.
5. Choose a **FROM email address** from the list of email addresses you have verified with Amazon SES in the **SES Region** you selected. To use an email address from a verified domain, configure email settings in the AWS Command Line Interface or the AWS API. For more information, see Verifying email addresses and domains in Amazon SES in the *Amazon Simple Email Service Developer Guide*.
6. Choose a **Configuration set** from the list of configuration sets in your chosen **SES Region**.
7. Enter a friendly **FROM sender name** for your email messages, in the format `John Stiles <johnstiles@example.com>`.
8. To customize the REPLY-TO email address, enter a valid email address in the **REPLY-TO email address** field.

Authorizing Amazon Cognito to send Amazon SES email on your behalf (from a custom FROM email address)

You can configure Amazon Cognito to send email from a custom FROM email address instead of its default address. To use a custom address, you must give Amazon Cognito permission to send email message from an Amazon SES verified identity. In most cases, you can grant permission by creating a sending authorization policy. For more information, see Using sending authorization with Amazon SES in the *Amazon Simple Email Service Developer Guide*.

When you configure a user pool to use Amazon SES for email messages, Amazon Cognito creates the `AWSServiceRoleForAmazonCognitoIdpEmailService` role in your account to grant access to Amazon SES. No sending authorization policy is needed when the `AWSServiceRoleForAmazonCognitoIdpEmailService` service-linked role is used. You only

need to add a sending authorization policy when you use both the default email functionality in your user pool *and* a verified Amazon SES identity as the FROM address.

For more information about the service-linked role that Amazon Cognito creates, see Using service-linked roles for Amazon Cognito.

The following example sending authorization policy grants Amazon Cognito a limited ability to use an Amazon SES verified identity. Amazon Cognito can only send email messages when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition. For more examples, see Amazon SES sending authorization policy examples in the *Amazon Simple Email Service Developer Guide*.

> **Note**
>
> In this example, the "Sid" value is an arbitrary string that uniquely identifies the statement. For more information about policy syntax, see Amazon SES sending authorization policies in the *Amazon Simple Email Service Developer Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "stmnt1234567891234",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "email.cognito-idp.amazonaws.com"
                ]
            },
            "Action": [
                "SES:SendEmail",
                "SES:SendRawEmail"
            ],
            "Resource": "arn:aws:ses:eusc-de-
east-1:111122223333:identity/support@example.com",
            "Condition": {
                "StringEquals": {
```

```
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/us-east-1_EXAMPLE"
                }
            }
        }
    ]
}
```

The Amazon Cognito console adds a similar policy for you when you select an Amazon SES identity from the drop-down menu. If you use the CLI or API to configure the user pool, you must attach a policy structured like the previous example to your Amazon SES Identity.

Creating user accounts as administrator

User pools aren't only a customer identity and access management (CIAM) user directory, where anyone on the internet can sign up for a user profile in your application. You can disable self-service sign-up. You might already know your customers and want to only admit those who have been authorized in advance. You can put manual authentication guardrails around your application with a private SAML 2.0 or OIDC identity provider, by importing users, by screening users at sign-up—or by creating users with administrative API operations. Your workflow for administrative creation of users can be programmatic, provisioning users after they register in another system, or it can be on a case-by-case or testing basis in the Amazon Cognito console.

When you create users as an administrator, Amazon Cognito sets a temporary password for them and sends a welcome, or invitation, message. They can follow the link in their invitation message and sign in for the first time, setting a password and confirming their account. The page that follows describes how to create new users and configure the welcome message. For more information about user creation with the user pools API and an AWS SDK or CDK, see AdminCreateUser.

> **Users with temporary passwords start in the FORCE_CHANGE_PASSWORD status**
>
> When you create a user with a temporary password, Amazon Cognito places the user in the FORCE_CHANGE_PASSWORD status. A user in this status can't sign in until they set a permanent password in response to a NEW_PASSWORD_REQUIRED challenge.

To move an administrator-created user to the `CONFIRMED` status without a first-time sign-in, call the AdminSetUserPassword API operation with `Permanent` set to `True`. This request sets a permanent password and confirms the user.

After you create your user pool, you can create users using the AWS Management Console, as well as the AWS Command Line Interface or the Amazon Cognito API. You can create a profile for a new user in a user pool and send a welcome message with sign-up instructions to the user via SMS or email.

The following are some examples of how administrators can manage users in user pools.

- Create a new user profile in the Amazon Cognito console or with the `AdminCreateUser` API operation.
- Make username-and-password, passwordless, passkey, and custom authentication flows available to your user pool and app client.
- Set user attribute values.
- Create custom attributes.
- Set the value of immutable custom attributes in `AdminCreateUser` API requests. This feature isn't available in the Amazon Cognito console.
- Specify a temporary password, create a user without a password, or allow Amazon Cognito to automatically generate a password.
- Create new users and automatically confirm their accounts, verify their email addresses, or verify their phone numbers.
- Specify custom SMS and email invitation messages for new users via the AWS Management Console or Lambda triggers like custom message, custom SMS sender, and custom email sender.
- Specify whether invitation messages are sent via SMS, email, or both.
- Resend the welcome message to an existing user by calling the `AdminCreateUser` API, specifying `RESEND` for the `MessageAction` parameter.
- Suppress the sending of the invitation message when the user is created.
- Specify an expiration time limit of up to 90 days for new user accounts.
- Allow users to sign themselves up or require that new users only be added by the administrator.

Administrators can also sign users in with AWS credentials in a server-side application. For more information, see Authorization models for API and SDK authentication.

User authentication flows and creating users

Administrative creation of users has options that differ based on the configuration of your user pool. The *authentication flows*, or methods available to users for sign-in and MFA, can change how you create users and the messages that you send to them. The following are some authentication flows that are available in user pools.

- Username and password
- Passkeys
- Sign-in with third-party IdPs
- Passwordless with email and SMS one-time passwords (OTPs)
- Multi-factor authentication with email, SMS, and authenticator-app OTPs
- Custom authentication with Lambda triggers

For more information about how to configure these sign-in factors, see Authentication with Amazon Cognito user pools.

Create users without passwords

If you have enabled passwordless sign-in for your user pool, you can create users without passwords. To create a user without a password, you must provide attribute values for an available passwordless sign-in factor. For example, if email OTP passwordless sign-in is available in your user pool, you can create a user with no password and an email address attribute. If the only authentication flows available to new users require a password, for example passkey or username-password, you must create or generate a temporary password for each new user.

To create a new user without a password

- Choose **Don't set a password** in the Amazon Cognito console
- Omit or leave blank the `TemporaryPassword` parameter of your `AdminCreateUser` API request

Users without passwords are automatically confirmed

Normally new users get a temporary password and go into a `FORCE_CHANGE_PASSWORD` status when you create them. When you create users without passwords, they immediately go into a `CONFIRMED` state. You can't resend confirmation codes to these users in the `CONFIRMED` state.

Invitation messages change for users without passwords.

By default, Amazon Cognito sends an invitation message to new users that says `Your username is {userName} and your password is {####}.` When you create users with no password, the message says `Your username is {userName}.` Customize your invitation message to reflect whether you will set passwords for users. Omit out the `{####}` password variable in passwordless authentication models.

You can't autogenerate passwords when passwordless factors are available

If you have configured your user pool to support email or phone OTP passwordless sign-in, you can't automatically generate a password. For each user who will have a password, you must set a temporary password when you create their profile.

Passwordless users must have values for all required attributes

When you create a user *without* a password, your request only succeeds if the user provides values for all attributes that you have marked as required in your user pool. This applies to any required attribute, not only the phone number and email attributes required for OTP delivery.

Creating users who will provide required-attribute values later

You might want to require attributes in your user pool but collect those attributes after you administratively create users, during user interaction in your application. Administrators can omit values for required attributes when they create users *with temporary passwords*. You can't omit required-attribute values for passwordless users.

Users with missing values for required attributes and a temporary password get a NEW_PASSWORD_REQUIRED challenge at first sign-in. They can then provide a value for the missing required attributes in the `requiredAttributes` parameter. You can create users with passwords and without required attributes only if all required attributes are mutable. Users can only complete sign-in with `NEW_PASSWORD_REQUIRED` challenges and required-attribute values if the required attributes are writeable from the app client they sign in with.

When you set a permanent password for an administrator-created user, their status changes to `CONFIRMED` and your user pool doesn't prompt them for a new password *or* required attributes at their first sign-in.

Creating a new user in the AWS Management Console

You can set user password requirements, configure the invitation and verification messages sent to users, and add new users with the Amazon Cognito console.

Set a password policy and enable self-registration

You can configure settings for minimum password complexity and whether users can sign up using public APIs in your user pool.

Configure a password policy

1. Navigate to the Amazon Cognito console, and choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Authentication methods** menu and locate **Password policy**. Choose **Edit**.
4. Choose a **Password policy mode** of **Custom**.
5. Choose a **Password minimum length**. For limits to the password length requirement, see User pools resource quotas.
6. Choose a **Password complexity** requirement.
7. Choose how long password set by administrators should be valid for.
8. Choose **Save changes**.

Allow self-service sign-up

1. Navigate to the Amazon Cognito console, and choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Sign-up** menu and locate **Self-service sign-up**. Select **Edit**.
4. Choose whether to **Enable self-registration**. Self-registration is typically used with public app clients that need to register new users in your user pool without distributing a client secret or AWS Identity and Access Management (IAM) API credentials.

> **Disabling self-registration**
>
> If you do not enable self-registration, new users must be created by administrative API actions using IAM API credentials or by sign-in with federated providers.

5. Choose **Save changes**.

Customize email and SMS messages

Customize user messages

You can customize the messages that Amazon Cognito sends to your users when you invite them to sign in, they sign up for a user account, or they sign in and are prompted for multi-factor authentication (MFA).

> **Note**
>
> An **Invitation message** is sent when you create a user in your user pool and invite them to sign in. Amazon Cognito sends initial sign-in information to the user's email address or phone number.
> A **Verification message** is sent when a user signs up for a user account in your user pool. Amazon Cognito sends a code to the user. When the user provides the code to Amazon Cognito, they verify their contact information and confirm their account for sign-in. Verification codes are valid for 24 hours.
> An **MFA message** is sent when you enable SMS MFA in your user pool, and a user that has configured SMS MFA signs in and is prompted for MFA.

1. Navigate to the Amazon Cognito console, and choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Message templates** menu and select **Verification message**, **Invitation message**, or **MFA message** and choose **Edit**.
4. Customize the messages for the chosen message type.

   > **Note**
   >
   > All variables in message templates must be included when you customize the message. If the variable, for example **{####}**, is not included, your user will have insufficient information to complete the message action.
   > For more information, see Message templates.

5. a. **Verification messages**

i. Choose a **Verification type** for **Email** messages. A **Code** verification sends a numeric code that the user must enter. A **Link** verification sends a link the user can click to verify their contact information. The text in the variable for a **Link** message is displayed as hyperlink text. For example, a message template using the variable {##Click here##} is displayed as Click here in the email message.

ii. Enter an **Email subject** for **Email** messages.

iii. Enter a custom **Email message** template for **Email** messages. You can customize this template with HTML.

iv. Enter a custom **SMS message** template for **SMS** messages.

v. Choose **Save changes**.

b. **Invitation messages**

i. Enter an **Email subject** for **Email** messages.

ii. Enter a custom **Email message** template for **Email** messages. You can customize this template with HTML.

iii. Enter a custom **SMS message** template for **SMS** messages.

iv. Choose **Save changes**.

c. **MFA messages**

i. Enter a custom **SMS message** template for **SMS** messages.

ii. Choose **Save changes**.

Create a user

Create a user

You can create new users for your user pool from the Amazon Cognito console. Typically, users can sign in after they set a password. To sign in with an email address, a user must verify the `email` attribute. To sign in with a phone number, the user must verify the `phone_number` attribute. To confirm accounts as an administrator, you can also use the AWS CLI or API, or create user profiles with a federated identity provider. For more information, see the Amazon Cognito API Reference.

1. Navigate to the Amazon Cognito console, and choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Users** menu, and choose **Create a user**.

4. Review the **User pool sign-in and security requirements** for guidance on password requirements, available account recovery methods, and alias attributes for your user pool.
5. Choose how you want to send an **Invitation message**. Choose SMS message, email message, or both. To suppress the invitation message, choose **Don't send an invitation**.

   > **Note**
   >
   > Before you can send invitation messages, configure a sender and an AWS Region with Amazon Simple Notification Service and Amazon Simple Email Service in the **Authentication methods** menu of your user pool . Recipient message and data rates apply. Amazon SES bills you for email messages separately, and Amazon SNS bills you for SMS messages separately.

6. Choose a **Username** for the new user.
7. Choose if you want to **Create a password** or have Amazon Cognito **Generate a password** for the user. The option to generate a password isn't available if passwordless sign-in is available in the user pool. Any temporary password must adhere to the user pool password policy.
8. Choose **Create**.
9. Choose the **Users** menu and choose the **User name** entry for the user. Add and edit **User attributes** and **Group memberships**. Review **User event history**.

Adding groups to a user pool

Support for groups in Amazon Cognito user pools enables you to create and manage groups, add users to groups, and remove users from groups. Use groups to create collections of users to manage their permissions or to represent different types of users. You can assign an AWS Identity and Access Management (IAM) role to a group to define the permissions for members of a group.

You can use groups to create a collection of users in a user pool, which is often done to set the permissions for those users. For example, you can create separate groups for users who are readers, contributors, and editors of your website and app. Using the IAM role associated with a group, you can also set different permissions for those different groups so that only contributors can put content into Amazon S3 and only editors can publish content through an API in Amazon API Gateway.

Amazon Cognito creates a user group for each OIDC, SAMl, and social identity provider (IdP) that you add to your user pool. The name of the group is in the format `[user pool ID]_[IdP`

name], for example us-east-1_EXAMPLE_MYSSO or us-east-1_EXAMPLE_Google. Each unique automatically-generated IdP user profile is automatically added to this group. Linked users aren't automatically added to this group, but you can add their profiles to the group in a separate process.

You can create and manage groups in a user pool from the AWS Management Console, the APIs, and the CLI. As a developer (using AWS credentials), you can create, read, update, delete, and list the groups for a user pool. You can also add users and remove users from groups.

There is no additional cost for using groups within a user pool. See Amazon Cognito Pricing for more information.

Assigning IAM roles to groups

You can use groups to control permissions to your resources using an IAM role. IAM roles include trust policies and permission policies. The role trust policy specifies who can use the role. The permissions policies specify the actions and resources that your group members can access. When you create an IAM role, set up the role trust policy to allow your group's users to assume the role. In the role permissions policies, specify the permissions that you want your group to have.

When you create a group in Amazon Cognito, you specify an IAM role by providing the role's ARN. When group members sign in using Amazon Cognito, they can receive temporary credentials from the identity pools. Their permissions are determined by the associated IAM role.

Individual users can be in multiple groups. As a developer, you have the following options for automatically choosing the IAM role when a user is in multiple groups:

- You can assign precedence values to each group. The group with the better (lower) precedence will be chosen and its associated IAM role will be applied.
- Your app can also choose from among the available roles when requesting AWS credentials for a user through an identity pool, by specifying a role ARN in the GetCredentialsForIdentity CustomRoleARN parameter. The specified IAM role must match a role that is available to the user.

Assigning precedence values to groups

A user can belong to more than one group. In the user's access and ID tokens, the cognito:groups claim contains the list of all the groups a user belongs to. The cognito:roles claim contains the list of roles corresponding to the groups.

Because a user can belong to more than one group, each group can be assigned a precedence. This is a non-negative number that specifies the precedence of this group relative to the other groups that a user belongs to in the user pool. Zero is the top precedence value. Groups with lower precedence values take precedence over groups with higher or null precedence values. If a user belongs to two or more groups, the group with the lowest precedence value will have its IAM role applied to the `cognito:preferred_role` claim in the user's ID token.

Two groups can have the same precedence value. If this happens, neither group takes precedence over the other. If two groups with the same precedence value have the same role ARN, that role is used in the `cognito:preferred_role` claim in ID tokens for users in each group. If the two groups have different role ARNs, the `cognito:preferred_role` claim is not set in users' ID tokens.

Using groups to control permission with Amazon API Gateway

You can use groups in a user pool to control permission with Amazon API Gateway. The groups that a user is a member of are included in both the ID token and access token from a user pool in the `cognito:groups` claim. You can submit ID or access tokens with requests to Amazon API Gateway and use an Amazon Cognito user pool authorizer for a REST API. For more information, see Control access to a REST API using Amazon Cognito user pools as authorizer in the API Gateway Developer Guide.

You can also authorize access to an Amazon API Gateway HTTP API with a custom JWT authorizer. For more information, see Controlling access to HTTP APIs with JWT authorizers in the API Gateway Developer Guide.

Limitations on groups

User groups are subject to the following limitations:

- The number of groups you can create is limited by the Amazon Cognito service quotas.
- Groups cannot be nested.
- You cannot search for users in a group.
- You cannot search for groups by name, but you can list groups.

Creating a new group in the AWS Management Console

Use the following procedure to create a new group.

To create a new group

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Groups** menu, and then choose **Create a group**.
5. On the **Create a group** page, in **Group name**, enter a friendly name for your new group.
6. You can optionally provide additional information about this group using any of the following fields:

   - **Description** - Enter details about what this new group will be used for.
   - **Precedence** - Amazon Cognito evaluates and applies all group permissions for a given user based on which groups that they belong to has a lower precedence value. The group with the lower precedence will be chosen and its associated IAM role will be applied. For more information, see Assigning precedence values to groups.
   - **IAM role** - You can assign an IAM role to your group when you need to control permissions to your resources. If you are integrating a user pool with an identity pool, the **IAM role** setting determines which role is assigned in the user's ID token if the identity pool is configured to choose the role from the token. For more information, see Assigning IAM roles to groups.
   - **Add users to this group** - Add existing users as members of this group after it is created.
7. Choose **Create** to confirm.

Managing and searching for user accounts

Users pools can contains millions of users. Working with a dataset of this size is a challenge for administrators. Amazon Cognito has tools for finding and modifying user profiles. The top methods for finding users are the **Users** menu of the Amazon Cognito console, and with ListUsers. Of the methods that retrieve information about users, these are the options that don't have a cost impact unlike, for example, AdminGetUser.

This section of the guide has information about finding and updating user profiles in a user pool.

Viewing user attributes

Use the following procedure to view user attributes in the Amazon Cognito console.

To view user attributes

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select a user in the list.
5. On the user details page, under **User attributes**, you can view which attributes are associated with the user.

Resetting a user's password

Use the following procedure to reset a user's password in the Amazon Cognito console.

To reset a user's password

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select a user in the list.
5. On the user details page, choose **Actions**, **Reset password**.
6. In the **Reset password** dialog, review the information and when ready, choose **Reset**.

   This action immediately results in a confirmation code being sent to the user and disables the user's current password by changing the user state to `RESET_REQUIRED`. The **Reset password** code is valid for 1 hour.

Enable, disable, and delete user accounts

You can delete unused user profiles or, if you want to temporarily prevent access, disable them. Users can delete their own accounts, but only user pool administrators can enable and disable user accounts.

Effect of deletion

Users can't sign in with deleted user accounts and to regain access, must sign up or be created again.

Effect of disabling accounts

When you disable a user account, Amazon Cognito automatically invalidates all authenticated sessions, deactivates the user account for sign-in, and revokes their access and refresh tokens. Amazon Cognito returns an `invalid_request` error with the message `User is not enabled` when a user tries to sign in to an account that you disabled. This behavior doesn't change with your user existence disclosure settings for the app client. You can disable local user accounts and the local profiles of federated user accounts. When users sign in with managed login or the classic hosted UI, then you disable their account, and then they try to sign in again with the the browser cookie that maintains their authenticated session, Amazon Cognito redirects them to the login page.

Effect of enabling accounts

Users can immediately sign in to accounts after you enable them. User accounts are enabled by default. Users' attributes and passwords remain the same as before their account was disabled. Tokens that your application revoked, whether you disabled the user account or separately revoked the refresh token, remain non-valid after you enable the user account that owned the token.

Delete a user account (console)

To delete a user account

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select the radio button next to the username of a user in the list.
5. Choose **Delete**.
6. Choose **Disable user access**.
7. Choose **Delete**.

Delete a user account (API)

Users can delete their accounts with the self-service access-token-authorized DeleteUser API operation. The following is an example `DeleteUser` request body.

```
{
```

```
   "AccessToken": "eyJra456defEXAMPLE"
}
```

Administrators can delete user accounts with the IAM-authorized AdminDeleteUser API operation. The following is an example `AdminDeleteUser` request body.

```
{
   "Username": "testuser",
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Disable a user account (console)

To disable a user account

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select the username of a user in the list.
5. On the user details page, choose **Actions**, **Disable user access**.
6. In the dialog that this creates, choose **Disable**.

Disable a user account (API)

Administrators can disable user accounts with the IAM-authorized AdminDisableUser API operation. The following is an example `AdminDisableUser` request body.

```
{
   "Username": "testuser",
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Enable a user account (console)

To enable a user account

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.

3. Choose an existing user pool from the list.
4. Choose the **Users** menu and select the username of a user in the list.
5. On the user details page, choose **Actions**, **Enable user access**.
6. In the dialog that this creates, choose **Enable**.

Enable a user account (API)

Administrators can enable user accounts with the IAM-authorized AdminEnableUser API operation. The following is an example `AdminEnableUser` request body.

```
{
   "Username": "testuser",
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Searching user attributes

If you have already created a user pool, you can search from the **Users** panel in the AWS Management Console. You can also use the Amazon Cognito ListUsers API, which accepts a **Filter** parameter.

You can search for any of the following standard attributes. Custom attributes are not searchable.

- username (case-sensitive)
- email
- phone_number
- name
- given_name
- family_name
- preferred_username
- cognito:user_status (called **Status** in the Console) (case-insensitive)
- status (called **Enabled** in the Console) (case-sensitive)
- sub

> **Note**
>
> You can also list users with a client-side filter. The server-side filter matches no more than 1 attribute. For advanced search, use a client-side filter with the `--query` parameter of the `list-users` action in the AWS Command Line Interface. When you use a client-side filter, ListUsers returns a paginated list of zero or more users. You can receive multiple pages in a row with zero results. Repeat the query with each pagination token that is returned until you receive a null pagination token value, then review the combined result.
> For more information about server-side and client-side filtering, see Filtering AWS CLI output in the AWS Command Line Interface User Guide.

Searching for users with the AWS Management Console

If you have already created a user pool, you can search from the **Users** panel in the AWS Management Console.

AWS Management Console searches are always prefix ("starts with") searches.

To search for a user in the Amazon Cognito console

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu and enter the username in the search field. Note that some attribute values are case-sensitive (for example, **Username**).

   You can also find users by adjusting the search filter to narrow the scope down to other user properties, such as **Email**, **Phone number**, or **Last name**.

Searching for users with the `ListUsers` API

To search for users from your app, use the Amazon Cognito ListUsers API. This API uses the following parameters:

- `AttributesToGet`: An array of strings, where each string is the name of a user attribute to be returned for each user in the search results. To retrieve all attributes, don't include an

`AttributesToGet` parameter or request `AttributesToGet` with a value of the literal string `null`.

- `Filter`: A filter string of the form "`AttributeName Filter-Type "AttributeValue"`". Quotation marks within the filter string must be escaped using the backslash (\) character. For example, `"family_name = \"Reddy\""`. If the filter string is empty, `ListUsers` returns all users in the user pool.
  - `AttributeName`: The name of the attribute to search for. You can only search for one attribute at a time.

    > **Note**
    >
    > You can only search for standard attributes. Custom attributes are not searchable. This is because only indexed attributes are searchable, and custom attributes cannot be indexed.

  - `Filter-Type`: For an exact match, use =, for example, `given_name = "Jon"`. For a prefix ("starts with") match, use ^=, for example, `given_name ^= "Jon"`.
  - `AttributeValue`: The attribute value that must be matched for each user.
- `Limit`: Maximum number of users to be returned.
- `PaginationToken`: A token to get more results from a previous search. Amazon Cognito expires the pagination token after one hour.
- `UserPoolId`: The user pool ID for the user pool on which the search should be performed.

All searches are case-insensitive. Search results are sorted by the attribute named by the `AttributeName` string, in ascending order.

Examples of using the `ListUsers` API

The following example returns all users and includes all attributes.

```
{
    "AttributesToGet": null,
    "Filter": "",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

The following example returns all users whose phone numbers start with "+1312" and includes all attributes.

```
{
    "AttributesToGet": null,
    "Filter": "phone_number ^= \"+1312\"",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

The following example returns the first 10 users whose family name is "Reddy". For each user, the search results include the user's given name, phone number, and email address. If there are more than 10 matching users in the user pool, the response includes a pagination token.

```
{
    "AttributesToGet": [
        "given_name",
        "phone_number",
        "email"
    ],
    "Filter": "family_name = \"Reddy\"",
    "Limit": 10,
    "UserPoolId": "us-east-1_samplepool"
}
```

If the previous example returns a pagination token, the following example returns the next 10 users that match the same filter string.

```
{
    "AttributesToGet": [
        "given_name",
        "phone_number",
        "email"
    ],
    "Filter": "family_name = \"Reddy\"",
    "Limit": 10,
    "PaginationToken": "pagination_token_from_previous_search",
```

```
    "UserPoolId": "us-east-1_samplepool"
}
```

Passwords, account recovery, and password policies

All users who sign in to a user pool, even federated users, have passwords assigned to their user profiles. Local users and linked users must provide a password when they sign in. Federated users don't use user pool passwords, but sign in with their identity provider (IdP). You can permit users to reset their own passwords, reset or change passwords as an administrator, and set policies for password complexity and history.

Amazon Cognito doesn't store user passwords in plaintext. Instead, it stores a hash of each user's password with a user-specific salt. Because of this, you can't retrieve existing passwords from the user profiles in your user pools. As a best practice, don't store plaintext user passwords anywhere. Perform password resets when users forget their passwords.

Password reset and recovery

Users forget their passwords. You might want them to be able to reset their password themselves, or you might want to require that an administrator resets their password for them. Amazon Cognito user pools have options for both models. This part of the guide covers the user pool settings and the API operations for password reset.

The ForgotPassword API operation and the managed login option **Forgot your password?** send users a code that, when they confirm that they have the correct code, gives them an opportunity to set a new password with ConfirmForgotPassword. This is the self-service password-recovery model.

Recovery of unverified users

You can send recovery messages to users who have verified their email address or phone number. If they don't have a confirmed recovery email or phone, a user pool administrator can mark their email address or phone number verified. Edit the user's **User attributes** in the Amazon Cognito console and select the checkbox next to **Mark phone number as verified** or **Mark email address as verified**. You can also set `email_verified` or `phone_number_verified` to true in an AdminUpdateUserAttributes request. For new users, the ResendConfirmationCode API operation sends a new code to their email address or phone number and they can complete self-service confirmation and verification.

Reset passwords as an administrator

The AdminSetUserPassword and AdminResetUserPassword API operations are the administrator-inititated methods of password reset. `AdminSetUserPassword` sets a temporary or permanent password, and `AdminResetUserPassword` sends users a password-reset code in the same way as `ForgotPassword`.

Configure password reset and recovery

Amazon Cognito automatically selects your account-recovery options from the required attributes and sign-in options that you choose when you create a user pool in the console. You can modify these default settings.

A user's preferred MFA method influences the methods they can use to recover their password. Users whose preferred MFA is by email message can't receive a password-reset code by email. Users whose preferred MFA is by SMS message can't receive a password-reset code by SMS.

Your password recovery settings must provide an alternative option when users aren't eligible for your preferred password-reset method. For example, your recovery mechanisms might have email as first priority and email MFA might be an option in your user pool. In this case, add SMS-message account recovery as a second option or use administrative API operations to reset passwords for those users.

Amazon Cognito replies to password-reset requests from users who don't have a valid recovery method with an `InvalidParameterException` error response.

> **Note**
>
> Users can't receive MFA and password reset codes at the same email address or phone number. If they use one-time passwords (OTPs) from email messages for MFA, they must use SMS messages for account recovery. If they use OTPs from SMS messages for MFA, they must use email messages for account recovery. In user pools with MFA, users might be unable to complete self-service password recovery if they have attributes for their email address but no phone number, or their phone number but no email address.
> To prevent the state where users can't reset their passwords in user pools with this configuration, set the `email` and phone_number attributes as required. As an alternative, you can set up processes that always collect and set those attributes when users sign up or when your administrators create user profiles. When users have both attributes, Amazon Cognito automatically sends password-reset codes to the destination that is *not* the user's MFA factor.

The following procedure configures self-service account recovery in a user pool.

Configure self-service password reset (API/SDK)

The `AccountRecoverySetting` parameter is the user pool parameter that sets the methods that users can use to recover their password in ForgotPassword API requests or when they select **Forgot password?** in managed login. `ForgotPassword` sends a recovery code to a verified email or a verified phone number. The recovery code is valid for one hour. When you specify an AccountRecoverySetting for your user pool, Amazon Cognito chooses the code delivery destination based on the priority that you set.

When you define `AccountRecoverySetting` and a user has SMS MFA configured, SMS cannot be used as an account recovery mechanism. The priority for this setting is determined with 1 being of the highest priority. Amazon Cognito sends a verification to only one of the specified methods. The following example `AccountRecoverySetting` sets email addresses as the primary destination for account-recovery codes, falling back to SMS message if the user doesn't have an email address attribute.

```
"AccountRecoverySetting": {
   "RecoveryMechanisms": [
      {
         "Name": "verified_email",
         "Priority": 1
      },
      {
         "Name": "verified_phone_number",
         "Priority": 2
      }
   ]
}
```

The value `admin_only` turns off self-service account recovery, instead requiring users to contact their administrator for password reset. You cannot use `admin_only` with any other account recovery mechanism. The following e

```
"AccountRecoverySetting": {
   "RecoveryMechanisms": [
      {
         "Name": "admin_only",
         "Priority": 1
```

```
        }
    ]
}
```

If you do not specify `AccountRecoverySetting`, Amazon Cognito sends the recovery code to a verified phone number first, and to a verified email address if users don't have a phone number attribute.

For more information about `AccountRecoverySetting`, see CreateUserPool and UpdateUserPool.

Configure self-service password reset (console)

Configure account-recovery and password-reset options from the **Sign-in** menu of your user pool.

To set up user account recovery

1. Sign in to the Amazon Cognito console.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Sign-in** menu. Locate **User account recovery** and choose **Edit**
5. To permit users to reset their own passwords, choose **Enable self-service account recovery**.
6. Configure the delivery method for the password-recovery codes that your user pool sends to users. Under **Delivery method for user account recovery messages**, select an available option. As a best practice, choose an option that has a secondary method for sending messages, for example **Email if available, otherwise SMS**. With a secondary delivery method, Amazon Cognito can send codes to users in a way that requires them to use a different medium for password reset than for MFA.
7. Select **Save changes**.

Forgot password behavior

In a given hour, we allow between 5 and 20 attempts for a user to request or enter a password reset code as part of forgot-password and confirm-forgot-password actions. The exact value depends on the risk parameters associated with the requests. Please note that this behavior is subject to change.

Adding user pool password requirements

Strong, complex passwords are a security best practice for your user pool. Especially in applications that are open to the internet, weak passwords can expose your users' credentials to systems that guess passwords and try to access your data. The more complex a password is, the more difficult it is to guess. Amazon Cognito has additional tools for security-conscious administrators, like threat protection and AWS WAF web ACLs, but your password policy is a central element of the security of your user directory.

Passwords for local users in Amazon Cognito user pools don't automatically expire. As a best practice, log the time, date, and metadata of user password resets in an external system. With an external log of password age, your application or a Lambda trigger can look up a user's password age and require a reset after a given period.

You can configure your user pool to require a minimum password complexity that conforms to your security standards. Complex passwords have a minimum length of at least eight characters. They also include a mix of uppercase, numeric, and special characters.

With the Essentials or Plus feature tiers, you can also set a policy for password reuse. You can prevent a user from resetting their password to a new password that matches their current password or any of up to 23 additional previous passwords, for a maximum total of 24.

> **Note**
>
> Amazon Cognito doesn't enforce your password policy when you import users with password hashes. Validate password strength in your source system before you generate hashes for import.

To set a user pool password policy

1. Create a user pool and navigate to the **Configure security requirements** step, or access an existing user pool and navigate to the **Authentication methods** menu.
2. Navigate to **Password policy**.
3. Choose a **Password policy mode**. **Cognito defaults** configures your user pool with the recommended minimum settings. You can also choose a **Custom** password policy.
4. Set a **Password minimum length**. All users must sign up or be created with a password whose length is greater than or equal to this value. You can set this minimum value as high as 99, but your users can set passwords up to 256 characters long.

5. Configure password complexity rules under **Password requirements**. Choose the character types–numbers, special characters, uppercase letters, and lowercase letters–that you want to require at least one of in each user's password.

   You can require at least one of the following characters in passwords. After Amazon Cognito verifies that passwords contain the minimum required characters, your users' passwords can contain additional characters of any type up to the maximum password length.

   - Uppercase and lowercase basic latin letters
   - Numbers
   - The following special characters.

     ```
     ^ $ * . [ ] { } ( ) ? " ! @ # % & / \ , > < ' : ; | _ ~ ` = + -
     ```

   - Non-leading, non-trailing space characters.
6. Set a value for **Temporary passwords set by administrators expire in**. After this period has passed, a new user that you created in the Amazon Cognito console or with `AdminCreateUser` can't sign in and set a new password. After they sign in with their temporary password, their user accounts never expire. To update the password duration in the Amazon Cognito user pools API, set a value for TemporaryPasswordValidityDays in your CreateUserPool or UpdateUserPool API request.
7. Set a value for **Prevent use of previous passwords**, if available. To use this feature, choose the Essentials or Plus feature tier in your user pool. The value of this parameter is the number of previous passwords that a new password is prevented from matching when a user resets their password.

To reset access for an expired user account, do one of the following:

- Send a new temporary password and reset the expiration period with an AdminCreateUser API request that has `MessageAction` set to `RESEND`.
- Delete the user profile and create a new one.
- Generate a new confirmation code in an AdminResetUserPassword API request.

Importing users into a user pool

There are two ways you can import or migrate users from your existing user directory or user database into Amazon Cognito user pools. You can migrate users when they sign-in using Amazon Cognito for the first time with a user migration Lambda trigger. With this approach, users can continue using their existing passwords and will not have to reset them after the migration to your user pool. Alternatively, you can migrate users in bulk by uploading a CSV file containing the user profile attributes for all users. The following sections describe both these approaches.

More resources

- Approaches for migrating users to Amazon Cognito user pools
- AWS re:Inforce 2023 - Migrating to Amazon Cognito

Topics

- Importing users with a user migration Lambda trigger
- Importing users into user pools from a CSV file

Importing users with a user migration Lambda trigger

With this approach, you can seamlessly migrate users from your existing user directory to user pools when a user signs in for the first time with your app or requests a password reset. Add a Migrate user Lambda trigger function to your user pool and it receives metadata about users who try to sign in, and returns user profile information from an external identity source. For details and example code for this Lambda trigger, including request and response parameters, see Migrate user Lambda trigger parameters.

Before you start to migrate users, create a user migration Lambda function in your AWS account, and set the Lambda function as the user migration trigger in your user pool. Add an authorization policy to your Lambda function that permits only the Amazon Cognito service account principal, `cognito-idp.amazonaws.com` to invoke the Lambda function, and only in the context of your own user pool. For more information, see Using resource-based policies for AWS Lambda (Lambda function policies).

Sign-in process

1. The user opens your app and signs in with the Amazon Cognito user pools API or through managed login. For more information about how to facilitate sign-in with Amazon Cognito APIs, see Integrating Amazon Cognito authentication and authorization with web and mobile apps.
2. Your app sends the user name and password to Amazon Cognito. If your app has a custom sign-in UI that you built with an AWS SDK, your app must use InitiateAuth or AdminInitiateAuth with the `USER_PASSWORD_AUTH` or `ADMIN_USER_PASSWORD_AUTH` flow. When your app uses one of these flows, the SDK sends the password to the server.

   > **Note**
   >
   > Before you add a user migration trigger, activate the `USER_PASSWORD_AUTH` or `ADMIN_USER_PASSWORD_AUTH` flow in the settings of your app client. You must use these flows instead of the default `USER_SRP_AUTH` flow. Amazon Cognito must send a password to your Lambda function so that it can verify your user's authentication in the other directory. An SRP obscures your user's password from your Lambda function.

3. Amazon Cognito checks if the submitted user name matches a user name or alias in the user pool. You can set the user's email address, phone number, or preferred user name as an alias in your user pool. If the user doesn't exist, Amazon Cognito sends parameters, including the user name and password, to your Migrate user Lambda trigger function.
4. Your Migrate user Lambda trigger function checks for or authenticates the user with your existing user directory or user database. The function returns user attributes that Amazon Cognito stores in the user's profile in the user pool. You can return a `username` parameter only if the submitted user name matches an alias attribute. If you want users to continue to use their existing passwords, your function sets the attribute `finalUserStatus` to `CONFIRMED` in the Lambda response. Your app must return all `"response"` parameters shown at Migrate user Lambda trigger parameters.

   > **Important**
   >
   > Do not log the entire request event object in your user migration Lambda code. This request event object includes the user's password. If you don't sanitize the logs, passwords appear in CloudWatch Logs.

5. Amazon Cognito creates the user profile in your user pool, and returns tokens to your app client.
6. Your app performs token intake, accepts the user authentication, and proceeds to the requested content.

After you migrate your users, use USER_SRP_AUTH for sign-in. The Secure Remote Password (SRP) protocol doesn't send the password across the network, and provides security benefits over the USER_PASSWORD_AUTH flow that you use during migration.

In case of errors during migration, including client device or network issues, your app receives error responses from the Amazon Cognito user pools API. When this happens, Amazon Cognito might or might not create the user account in your user pool. The user should then attempt to sign in again. If sign-in fails repeatedly, attempt to reset the user's password with the forgot-password flow in your app.

The forgot-password flow also invokes your Migrate user Lambda trigger function with a UserMigration_ForgotPassword event source. Because the user doesn't submit a password when they request a password reset, Amazon Cognito doesn't include a password in the event that it sends to your Lambda function. Your function can only look up the user in your existing user directory and return attributes to add to the user profile in your user pool. After your function completes its invocation and returns its response to Amazon Cognito, your user pool sends a password reset code by email or SMS. In your app, prompt your user for their confirmation code and a new password, then send that information to Amazon Cognito in a ConfirmForgotPassword API request. You can also use the built-in pages for the forgot-password flow in managed login.

Additional resources

- Approaches for migrating users to Amazon Cognito user pools

Importing users into user pools from a CSV file

When you have an external identity store and the time to prepare your user pool for new local users, a bulk user import from a comma-separated values (CSV) file can be a low-effort, low-cost option for a migration to an Amazon Cognito user pool. A CSV file import is a process of downloading and populating a template file, then handing off the file to your user pool in an import job. You can use a CSV import to quickly create test users. You can also programmatically populate the file with read API requests to your external identity store, followed by parsing their details and attributes into write operations to the file.

By default, the import process sets values for all user attributes except **password**. This means that your users must change their passwords the first time they sign in. Your users are in a RESET_REQUIRED state when imported using this method.

Alternatively, you can import users with their existing password hashes. When you specify a password hashing algorithm during import job creation and include password hashes in your CSV file, Amazon Cognito imports the users with their existing passwords. These users are created with a `CONFIRMED` status and can sign in immediately without resetting their passwords. For more information, see Importing users with password hashes.

The lowest-effort way to import users from a CSV is to activate passwordless sign-in in your user pool. With email address and phone number attributes and the right user pool configuration, users can sign in with email or SMS one-time passwords (OTPs) immediately after your import job completes. For more information, see Requiring imported users to reset their passwords.

You can also set your users' passwords with an AdminSetUserPassword API request that sets the `Permanent` parameter to `true`. CSV import doesn't contribute to the billed monthly active users (MAUs) in your user pool. However, password-reset operations do generate MAUs. To manage costs when you import large numbers of users with password who might not be immediately active, set up your application to prompt users for a new password when they sign in and receive the `RESET_REQUIRED` challenge.

> **Note**
>
> The creation date for each user is the time when that user was imported into the user pool. Creation date is not one of the imported attributes.

Steps to create a user import job

1. Create an Amazon CloudWatch Logs role in the AWS Identity and Access Management (IAM) console.
2. Create the user import .csv file.
3. Create and run the user import job. Optionally, specify a password hashing algorithm to import users with their existing password hashes.
4. Upload the user import .csv file.
5. Start and run the user import job.
6. Use CloudWatch to check the event log.
7. If you didn't import password hashes, require the imported users to reset their passwords.

More resources

- Cognito User Profiles Export Reference Architecture for exporting user accounts between user pools

Topics

- Creating the CloudWatch Logs IAM role
- Creating the user import CSV file
- Creating and running the Amazon Cognito user pool import job
- Viewing the user pool import results in the CloudWatch console
- Requiring imported users to reset their passwords
- Importing users with password hashes

Creating the CloudWatch Logs IAM role

If you're using the Amazon Cognito CLI or API, then you need to create a CloudWatch IAM role. The following procedure describes how to create an IAM role that Amazon Cognito can use to write the results of your import job to CloudWatch Logs.

> **Note**
>
> When you create an import job in the Amazon Cognito console, you can create the IAM role at the same time. When you choose to **Create a new IAM role**, Amazon Cognito automatically applies the appropriate trust policy and IAM policy to the role.

To create the CloudWatch Logs IAM role for user pool import (AWS CLI, API)

1. Sign in to the AWS Management Console and open the IAM console at https://eusc-de-east-1.console.amazonaws-eusc.eu/iam/.
2. Create a new IAM role for an AWS service. For detailed instructions, see Creating a role for an AWS service in the *AWS Identity and Access Management User Guide*.
    a. When you select a **Use case** for your **Trusted entity type**, choose any service. Amazon Cognito isn't currently listed in service use cases.

b. In the **Add permissions** screen, choose **Create policy** and insert the following policy statement. Replace *REGION* with the AWS Region of your user pool, for example `us-east-1`. Replace *ACCOUNT* with your AWS account ID, for example 111122223333.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogGroup",
                "logs:CreateLogStream",
                "logs:DescribeLogStreams",
                "logs:PutLogEvents"
            ],
            "Resource": [
                "arn:aws-eusc:logs:eusc-de-east-1:111122223333:log-
group:/aws/cognito/*"
            ]
        }
    ]
}
```

3. Because you didn't choose Amazon Cognito as the trusted entity when you created the role, you now must manually edit the trust relationship of the role. Choose **Roles** from navigation pane of the IAM console, then choose the new role that you created.
4. Choose the **Trust relationships** tab.
5. Choose **Edit trust policy**.
6. Paste the following policy statement into **Edit trust policy**, replacing any existing text:

JSON

```
{
        "Version":"2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
```

```
                "Principal": {
                    "Service": "cognito-idp.amazonaws.com"
                },
                "Action": "sts:AssumeRole"
            }
        ]
    }
```

7. Choose **Update policy**.
8. Note the role ARN. You'll provide the ARN when you create your import job.

Alternatively, you can create the same role with the AWS CLI. The following commands create a role that Amazon Cognito can assume, attach a policy that permits writing to CloudWatch Logs, and return the role ARN. Replace *REGION* with the AWS Region of your user pool and *ACCOUNT* with your AWS account ID.

To create the CloudWatch Logs IAM role for user pool import (AWS CLI)

1. Create the role with a trust policy that allows Amazon Cognito to assume it.

```
aws iam create-role \
    --role-name CognitoUserImportRole \
    --assume-role-policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
                "Effect": "Allow",
                "Principal": { "Service": "cognito-idp.amazonaws.com" },
                "Action": "sts:AssumeRole"
            }
        ]
    }'
```

2. Attach a policy that permits the role to write import results to CloudWatch Logs.

```
aws iam put-role-policy \
    --role-name CognitoUserImportRole \
    --policy-name CognitoUserImportLogsPolicy \
    --policy-document '{
        "Version": "2012-10-17",
        "Statement": [
            {
```

```
                "Effect": "Allow",
                "Action": [
                    "logs:CreateLogGroup",
                    "logs:CreateLogStream",
                    "logs:PutLogEvents",
                    "logs:DescribeLogStreams"
                ],
                "Resource": "arn:aws:logs:REGION:ACCOUNT:log-group:/aws/cognito/*"
            }
        ]
    }'
```

3. Retrieve the role ARN to use as *ROLE_ARN* when you create your import job.

```
aws iam get-role \
    --role-name CognitoUserImportRole \
    --query 'Role.Arn' --output text
```

Creating the user import CSV file

Before you can import your existing users into your user pool, you must create a comma-separated values (CSV) file that contains the users that you want to import, and their attributes. From your user pool, you can retrieve a user import file with headers that reflect the attribute schema of your user pool. You can then insert user information that matches the formatting requirements in Formatting the CSV file.

Downloading the CSV file header (console)

Use the following procedure to download the CSV header file.

To download the CSV file header

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu.
5. In the **Import users** section, choose **Create an import job**.
6. Under **Upload CSV**, select the *template.csv* link and download the CSV file.

Downloading the CSV file header (AWS CLI)

To get a list of the correct headers, from the **Users menu** under **Import users**, select **Create import job**. In the dialog that follows, select the `template.csv` link to download a template file with your user pool attributes.

You can also run the following CLI command, where *USER_POOL_ID* is the user pool identifier for the user pool you'll import users into:

```
aws cognito-idp get-csv-header --user-pool-id "USER_POOL_ID"
```

Sample response:

```
{
    "CSVHeader": [
        "name",
        "given_name",
        "family_name",
        "middle_name",
        "nickname",
        "preferred_username",
        "profile",
        "picture",
        "website",
        "email",
        "email_verified",
        "gender",
        "birthdate",
        "zoneinfo",
        "locale",
        "phone_number",
        "phone_number_verified",
        "address",
        "updated_at",
        "cognito:mfa_enabled",
        "cognito:username"
    ],
    "UserPoolId": "USER_POOL_ID"
}
```

Formatting the CSV file

The downloaded user import CSV header file looks like the following string. It also includes any custom attributes you have added to your user pool.

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,pi
```

Edit your CSV file so that it includes this header and the attribute values for your users, and is formatted according to the following rules:

> **Note**
>
> For more information about attribute values, such as proper format for phone numbers, see Working with user attributes.

- The first row in the file is the downloaded header row, which contains the user attribute names.
- The order of columns in the CSV file doesn't matter.
- Each row after the first row contains the attribute values for a user.
- All columns in the header must be present, but you don't need to provide values in every column.
- The following attributes are required:
  - **cognito:username**
  - **email_verified** or **phone_number_verified**
    - At least one of the auto-verified attributes must be `true` for each user. An auto-verified attribute is an email address or phone number that Amazon Cognito automatically sends a code to when a new user joins your user pool.
    - The user pool must have at least one auto-verified attribute, either **email_verified** or **phone_number_verified**. If the user pool has no auto-verified attributes, the import job will not start.
    - If the user pool only has one auto-verified attribute, that attribute must be verified for each user. For example, if the user pool has only **phone_number** as an auto-verified attribute, the **phone_number_verified** value must be `true` for each user.

> **Note**
>
> For users to reset their passwords, they must have a verified email or phone number. Amazon Cognito sends a message containing a reset password code to the email or phone number specified in the CSV file. If the message is sent to the phone number, it is sent by SMS message. For more information, see Verifying contact information at sign-up.

  - **email** (if **email_verified** is `true`)
  - **phone_number** (if **phone_number_verified** is `true`)
  - Any attributes that you marked as required when you created the user pool
- Attribute values that are strings should *not* be in quotation marks.
- If an attribute value contains a comma, you must put a backslash (\) before the comma. This is because the fields in a CSV file are separated by commas.
- The CSV file contents should be in UTF-8 format without byte order mark.
- The **cognito:username** field is required and must be unique within your user pool. It can be any Unicode string. However, it cannot contain spaces or tabs.
- The **birthdate** values, if present, must be in the format *mm/dd/yyyy*. This means, for example, that a birthdate of February 1, 1985 must be encoded as **02/01/1985**.
- The **cognito:mfa_enabled** field must correspond to the MFA requirements of your user pool. If you've set multi-factor authentication (MFA) to be required in your user pool, this field must be `true` or blank for all users. If you've set MFA to be off, this field must be `false` or blank for all users. A blank value sets imported users' MFA-enabled status to the state required by the user pool. You can import users in an MFA-required user pool without a valid MFA factor, regardless of whether you set a `cognito:mfa_enabled` value. Users in this state have MFA active but can't sign in until they configure an email attribute, phone number attribute, or a TOTP, and that configuration is a valid MFA factor in your user pool.
- The maximum row length is 16,000 characters.
- The maximum CSV file size is 100 MB.
- The maximum number of rows (users) in the file is 500,000. This maximum doesn't include the header row.
- The **updated_at** field value is expected to be epoch time in seconds, for example: **1471453471**.
- Any leading or trailing white space in an attribute value will be trimmed.

The following list is a example CSV import file for a user pool with no custom attributes. Your user pool schema might differ from this example. In that case, you must provide test values in the CSV template that you download from your user pool.

```
cognito:username,name,given_name,family_name,middle_name,nickname,preferred_username,profile,pi
John,,John,Doe,,,,,,,johndoe@example.com,TRUE,,02/01/1985,,,+12345550100,TRUE,123 Any
 Street,,FALSE
Jane,,Jane,Roe,,,,,,,janeroe@example.com,TRUE,,01/01/1985,,,+12345550199,TRUE,100 Main
 Street,,FALSE
```

Creating and running the Amazon Cognito user pool import job

This section describes how to create and run the user pool import job by using the Amazon Cognito console and the AWS Command Line Interface (AWS CLI).

Topics

- Importing users from a CSV file (console)
- Importing users (AWS CLI)

Importing users from a CSV file (console)

The following procedure describes how to import the users from the CSV file.

To import users from the CSV file (console)

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Users** menu.
5. In the **Import users** section, choose **Create an import job**.
6. On the **Create import job** page, enter a **Job name**.
7. Choose to **Create a new IAM role** or to **Use an existing IAM role**.
    a. If you chose **Create a new IAM role**, enter a name for your new role. Amazon Cognito will automatically create a role with the correct permissions and trust relationship. The IAM principal that creates the import job must have permissions to create IAM roles.

   b. If you chose **Use an existing IAM role**, choose a role from the list under **IAM role selection**. This role must have the permissions and trust policy described in Creating the CloudWatch Logs IAM role.

8. Under **Upload CSV**, choose **Choose file** and attach the CSV file that you prepared.
9. Choose **Create job** to submit your job, but start it later. Choose **Create and start job** to submit your job and start it immediately.
10. If you created your job but didn't start it, you can start it later. In the **Users** menu under **Import users**, choose your import job, then select **Start**. You can also submit a StartUserImportJob API request from an AWS SDK.
11. Monitor the progress of your user import job in the **Users** menu under **Import users**. If your job doesn't succeed, you can select the **Status** value. For additional details, select **View the CloudWatch logs for more details** and review any issues in the CloudWatch Logs console.

Importing users (AWS CLI)

The following CLI commands are available for importing users into a user pool:

- `create-user-import-job`
- `get-csv-header`
- `describe-user-import-job`
- `list-user-import-jobs`
- `start-user-import-job`
- `stop-user-import-job`

To get the list of command line options for these commands, use the `help` command line option. For example:

```
aws cognito-idp get-csv-header help
```

Creating a user import job

After you create your CSV file, create a user import job by running the following CLI command, where *JOB_NAME* is the name you're choosing for the job, *USER_POOL_ID* is the user pool ID for the user pool into which the new users will be added, and *ROLE_ARN* is the role ARN you received in Creating the CloudWatch Logs IAM role:

```
aws cognito-idp create-user-import-job --job-name "JOB_NAME" --user-pool-id
 "USER_POOL_ID" --cloud-watch-logs-role-arn "ROLE_ARN"
```

The *PRE_SIGNED_URL* returned in the response is valid for 15 minutes. After that time, it will expire and you must create a new user import job to get a new URL.

Example response:

```
{
    "UserImportJob": {
        "Status": "Created",
        "SkippedUsers": 0,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl": "PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

Status values for a user import job

In the responses to your user import commands, you'll see one of the following `Status` values:

- `Created` - The job was created but not started.
- `Pending` - A transition state. You have started the job, but it has not begun importing users yet.
- `InProgress` - The job has started, and users are being imported.
- `Stopping` - You have stopped the job, but the job has not stopped importing users yet.
- `Stopped` - You have stopped the job, and the job has stopped importing users.
- `Succeeded` - The job has completed successfully.
- `Failed` - The job has stopped due to an error.
- `Expired` - You created a job, but did not start the job within 24-48 hours. All data associated with the job was deleted, and the job can't be started.

Uploading the CSV file

Use the following `curl` command to upload the CSV file containing your user data to the presigned URL that you obtained from the response of the `create-user-import-job` command.

```
curl -v -T "PATH_TO_CSV_FILE" -H "x-amz-server-side-encryption:aws:kms"
 "PRE_SIGNED_URL"
```

In the output of this command, look for the phrase `"We are completely uploaded and fine"`. This phrase indicates that the file was uploaded successfully. Your user pools don't keep the information in your import files after you run your import jobs. After they complete or expire, Amazon Cognito deletes your uploaded CSV file.

Describing a user import job

To get a description of your user import job, use the following command, where *USER_POOL_ID* is your user pool ID, and *JOB_ID* is the job ID that was returned when you created the user import job.

```
aws cognito-idp describe-user-import-job --user-pool-id "USER_POOL_ID" --job-id
 "JOB_ID"
```

Example Sample response:

```
{
    "UserImportJob": {
        "Status": "Created",
        "SkippedUsers": 0,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl": "PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn":"ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

In the preceding sample output, the *PRE_SIGNED_URL* is the URL that you uploaded the CSV file to. The *ROLE_ARN* is the CloudWatch Logs role ARN that you received when you created the role.

Listing your user import jobs

To list your user import jobs, use the following command:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 2
```

Example Sample response:

```
{
    "UserImportJobs": [
        {
            "Status": "Created",
            "SkippedUsers": 0,
            "UserPoolId": "USER_POOL_ID",
            "ImportedUsers": 0,
            "JobName": "JOB_NAME",
            "JobId": "JOB_ID",
            "PreSignedUrl":"PRE_SIGNED_URL",
            "CloudWatchLogsRoleArn":"ROLE_ARN",
            "FailedUsers": 0,
            "CreationDate": 1470957431.965
        },
        {
            "CompletionDate": 1470954227.701,
            "StartDate": 1470954226.086,
            "Status": "Failed",
            "UserPoolId": "USER_POOL_ID",
            "ImportedUsers": 0,
            "SkippedUsers": 0,
            "JobName": "JOB_NAME",
            "CompletionMessage": "Too many users have failed or been skipped during the
 import.",
            "JobId": "JOB_ID",
            "PreSignedUrl":"PRE_SIGNED_URL",
            "CloudWatchLogsRoleArn":"ROLE_ARN",
            "FailedUsers": 5,
            "CreationDate": 1470953929.313
        }
    ],
    "PaginationToken": "PAGINATION_TOKEN"
}
```

Jobs are listed in chronological order from last created to first created. The *PAGINATION_TOKEN* string after the second job indicates that there are additional results for this list command. To list the additional results, use the `--pagination-token` option as follows:

```
aws cognito-idp list-user-import-jobs --user-pool-id "USER_POOL_ID" --max-results 10 --
pagination-token "PAGINATION_TOKEN"
```

Starting a user import job

To start a user import job, use the following command:

```
aws cognito-idp start-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Only one import job can be active at a time per account.

Example Sample response:

```
{
    "UserImportJob": {
        "Status": "Pending",
        "StartDate": 1470957851.483,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "SkippedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl":"PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

Stopping a user import job

To stop a user import job while it is in progress, use the following command. After you stop the job, it cannot be restarted.

```
aws cognito-idp stop-user-import-job --user-pool-id "USER_POOL_ID" --job-id "JOB_ID"
```

Example Sample response:

```
{
    "UserImportJob": {
        "CompletionDate": 1470958050.571,
        "StartDate": 1470958047.797,
        "Status": "Stopped",
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "SkippedUsers": 0,
        "JobName": "JOB_NAME",
        "CompletionMessage": "The Import Job was stopped by the developer.",
        "JobId": "JOB_ID",
        "PreSignedUrl":"PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "FailedUsers": 0,
        "CreationDate": 1470957972.387
    }
}
```

Viewing the user pool import results in the CloudWatch console

You can view the results of your import job in the Amazon CloudWatch console.

Topics

- Viewing the results
- Interpreting the results

Viewing the results

The following steps describe how to view the user pool import results.

To view the results of the user pool import

1. Sign in to the AWS Management Console and open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. Choose **Logs**.
3. Choose the log group for your user pool import jobs. The log group name is in the form `/aws/cognito/userpools/`*USER_POOL_ID*`/`*USER_POOL_NAME*.

4. Choose the log for the user import job you just ran. The log name is in the form *JOB_ID*/*JOB_NAME*. The results in the log refer to your users by line number. No user data is written to the log. For each user, a line similar to the following appears:

   - `[SUCCEEDED] Line Number 5956 - The import succeeded.`
   - `[SKIPPED] Line Number 5956 - The user already exists.`
   - `[FAILED] Line Number 5956 - The User Record does not set any of the auto verified attributes to true. (Example: email_verified to true).`

Interpreting the results

Successfully imported users have their status set to "PasswordReset".

In the following cases, the user will not be imported, but the import job will continue:

- No auto-verified attributes are set to `true`.
- The user data doesn't match the schema.
- The user couldn't be imported due to an internal error.

In the following cases, the import job will fail:

- The Amazon CloudWatch Logs role cannot be assumed, doesn't have the correct access policy, or has been deleted.
- The user pool has been deleted.
- Amazon Cognito is unable to parse the .csv file.

Requiring imported users to reset their passwords

If your user pool only offers password-based sign-in, users must reset their passwords after they are imported. The first time they sign in, they can enter *any* password. Amazon Cognito prompts them to enter a new password in the API response to the sign-in request from your application.

If your user pool has passwordless authentication factors, Amazon Cognito defaults to those for imported users. They're not prompted for a new password, and can sign in immediately with a passwordless email or SMS OTP. You can also prompt users to set a password so that they can complete other sign-in methods like username-password and passkey. The following conditions apply to passwordless sign-in after user import.

1. You must import users with an attribute that corresponds to an available passwordless sign-in factor. If users can sign in with an email address, you must import an `email` attribute. If a phone number, you must import a `phone_number` attribute. If both, import a value for either attribute.
2. Normally, users import in a `RESET_REQUIRED` state where they must reset their password. If they are imported with the ability to sign in with a passwordless factor, Amazon Cognito sets their state to `CONFIRMED`.

For more information about passwordless authentication including how to set it up and how to construct the authentication flow in your application, see Authentication with Amazon Cognito user pools.

The following procedure describes the user experience in a custom-built login mechanism with local users in a `RESET_REQUIRED` after you import a CSV file. If your users sign in with managed login, instruct them to select the **Forgot password?** option, provide the code from their email or text message, and set a password.

Requiring imported users to reset their passwords

1. In your app, silently attempt sign-in for the current user with `InitiateAuth` using a random password.
2. Amazon Cognito returns a `NotAuthorizedException` when `PreventUserExistenceErrors` is enabled. Otherwise, it returns `PasswordResetRequiredException`.
3. Your app makes a `ForgotPassword` API request and resets the user's password.
    a. The app submits the username in a `ForgotPassword` API request.
    b. Amazon Cognito sends a code to the verified email or phone number. The destination depends on the values you provided for `email_verified` and `phone_number_verified` in your CSV file. The response to the `ForgotPassword` request indicates the destination of the code.

> **Note**
>
> Your user pool must be configured to verify emails or phone numbers. For more information, see Signing up and confirming user accounts.

c. Your app displays a message to your user to check the location where the code was sent, and prompts your user to enter the code and a new password.

d. The user enters the code and new password in the app.

e. The app submits the code and new password in a `ConfirmForgotPassword` API request.

f. Your app redirects your user to sign-in.

Importing users with password hashes

When you migrate users from an existing authentication system to Amazon Cognito, you can import their password hashes along with their user attributes. Your users can then sign in immediately with their existing passwords, without a password reset. After users sign in for the first time, Amazon Cognito transparently migrates their credentials to the native Amazon Cognito authentication system.

> **Important**
>
> Password hash import is not available for all user pools at this time. Password hash import requires the modern Amazon Cognito infrastructure with enhanced capabilities and scalability. Some user pools are still on a previous infrastructure and AWS will upgrade them to the new infrastructure, which unlocks this feature. In the Amazon Cognito console, eligible user pools display the password hashing algorithm option when creating import jobs, and ineligible pools do not display this option. For more information, see Amazon Cognito unlocks advanced capabilities with next-generation infrastructure in the AWS Security Blog.

How password hash import works

With password hash import, you can migrate users while preserving their ability to sign in with their existing passwords. The import process works as follows:

1. You create an import job and specify the password hashing algorithm that your source system used to create the password hashes.
2. You include the `password_hash` column in your CSV file with each user's password hash value.
3. Amazon Cognito imports users with their password hashes and creates them with a `CONFIRMED` status. Users can sign in immediately.

4. When a user signs in for the first time, Amazon Cognito verifies their password against the imported hash. After successful verification, Amazon Cognito migrates the user's credentials to the native authentication system.
5. All subsequent sign-ins use the native Amazon Cognito authentication system.

> **Important**
>
> All password hashes in a single import job must use the same algorithm. You specify the algorithm at the job level when you create the import job.

> **Note**
>
> Until a user completes their first sign-in and Amazon Cognito migrates their credentials, you can't use Secure Remote Password (SRP) authentication for that user. Use `USER_PASSWORD_AUTH` or `ADMIN_USER_PASSWORD_AUTH` flows for users with imported password hashes who haven't yet signed in.

Supported password hashing algorithms

Amazon Cognito supports the following password hashing algorithms for import:

BCRYPT

The bcrypt adaptive hash function. Amazon Cognito extracts all required parameters (salt, cost factor) from the hash string.

Format: `$2<a/b/x/y>$[cost]$[22-char salt][31-char hash]`

Example: `$2b$10$CtA.Rcu/szzn9U00wpUjOuN3vrgJRZycv4aOzcP3GzqzO8UDPEFq6`

Maximum cost factor: 12

SCRYPT

The scrypt password-based key derivation function. Amazon Cognito extracts all required parameters from the hash string.

Format: `N$r$p$hexSalt$hexHash`

Example: `65536$8$1$304dbaef7c5e828dc19c98f0600d18fe$4f69c498c12cd102d057356facf8d77e8d42407090491ea32c5b038f5a18c099`

Maximum parameters: N (CPU/memory cost) = 65536, r (block size) = 8, p (parallelism) = 1

`ARGON2ID`

The Argon2id password hashing algorithm. Amazon Cognito extracts all required parameters from the hash string.

Format: `$argon2id$v=N$m=M,t=T,p=P$salt$hash`

Example: `$argon2id$v=19$m=19456,t=2,p=1$ko/G5o1ms+ML08P95sQ8DA$AkVbvWSOqz7Hs3qthhWKxicOWnGLN+MBmpwc3emi5VA`

Maximum parameters: m (memory in KiB) = 19456, t (iterations) = 2, p (parallelism) = 1

`PBKDF2_SHA256`

Password-Based Key Derivation Function 2 with SHA-256. Amazon Cognito extracts all required parameters from the hash string.

Format: `$pbkdf2-sha256$iterations$salt$hash`

Example: `$pbkdf2-sha256$600000$1XZlmwLQ2hhM3JYuCPiArQ$Pfheg9Zi/v5lXU4yyLA0WFUYEd/rlaVbzrM9oMD6IrA`

Maximum iterations: 600000

> **Note**
>
> All supported algorithms are self-describing, which means Amazon Cognito can extract all required parameters (such as salt, cost factor, and iterations) directly from the hash string. You only need to specify the algorithm name when you create the import job.

If your password hash has parameter values that exceed the maximum bounds listed above, the import fails for that user. Review your source system's password hashing configuration before importing to ensure compatibility.

Adding password hashes to your CSV file

When you download the CSV template for user import, the template includes a `password_hash` column. Populate this column with the password hash values for users you want to import with their existing passwords.

CSV formatting rules for password hash import

- The `password_hash` column is required when your import job specifies a password hashing algorithm. If you leave the value empty for a specific user, Amazon Cognito imports that user without a password and sets the user to the `RESET_REQUIRED` state.
- If your import job specifies a password hashing algorithm but your CSV file doesn't include a `password_hash` column, the job fails.
- If a password hash value is malformed or doesn't match the expected format for the specified algorithm, the import fails for that user. Amazon Cognito doesn't create the user and records the failure in your Amazon CloudWatch Logs logs.
- Password hash values are case-sensitive. Make sure they match the exact format that the algorithm expects.

Example Sample CSV with password hashes

The following example shows a CSV file that imports users with bcrypt password hashes:

```
cognito:username,email,email_verified,phone_number,phone_number_verified,password_hash
alejandro_rosalez,alejandro_rosalez@example.com,TRUE,+12345550100,TRUE,$2b$10$CtA.Rcu/
szzn9U00wpUjOuN3vrgJRZycv4aOzcP3GzqzO8UDPEFq6
mary_major,mary_major@example.com,TRUE,+12345550199,TRUE,$2b$10$CtA.Rcu/
szzn9U00wpUjOuN3vrgJRZycv4aOzcP3GzqzO8UDPEFq6
```

Creating an import job with password hashes (AWS CLI)

To import users with password hashes, you must specify the password hashing algorithm when you create the import job. All users in the import job must use the same algorithm.

Use the following AWS Command Line Interface command to create an import job with password hashes. The `--password-hashing-algorithm` parameter specifies the algorithm used to create the password hashes in your CSV file.

```
aws cognito-idp create-user-import-job \
```

```
    --job-name "JOB_NAME" \
    --user-pool-id "USER_POOL_ID" \
    --cloud-watch-logs-role-arn "ROLE_ARN" \
    --password-hashing-algorithm BCRYPT
```

Replace *BCRYPT* with one of the supported algorithms: BCRYPT, SCRYPT, ARGON2ID, or PBKDF2_SHA256.

Example Sample response

```
{
    "UserImportJob": {
        "Status": "Created",
        "SkippedUsers": 0,
        "UserPoolId": "USER_POOL_ID",
        "ImportedUsers": 0,
        "JobName": "JOB_NAME",
        "JobId": "JOB_ID",
        "PreSignedUrl": "PRE_SIGNED_URL",
        "CloudWatchLogsRoleArn": "ROLE_ARN",
        "PasswordHashingAlgorithm": "BCRYPT",
        "FailedUsers": 0,
        "CreationDate": 1470957431.965
    }
}
```

The response includes the `PasswordHashingAlgorithm` field confirming the algorithm you specified. Upload your CSV file to the presigned URL and start the job as described in Uploading the CSV file.

User experience after password hash import

Users imported with password hashes have the following experience:

- **Immediate sign-in:** Users can sign in with their existing passwords immediately after import. Users don't receive a prompt to reset their passwords.
- **User status:** Amazon Cognito creates these users with a `CONFIRMED` status instead of `RESET_REQUIRED`.
- **Transparent migration:** When users sign in for the first time, Amazon Cognito verifies their password against the imported hash. After successful verification, Amazon Cognito migrates

their credentials to the native authentication system. All subsequent sign-ins use the native system.

- **Initial sign-in latency:** The first sign-in for users with imported password hashes might take slightly longer than subsequent sign-ins. Amazon Cognito must verify the password against the imported hash and migrate the credentials to the native authentication system.
- **Password reset:** If a user resets their password before their first sign-in, their imported password hash is replaced with a new password using the native Amazon Cognito authentication system.

Troubleshooting password hash import errors

If a password hash import fails for a user, Amazon Cognito records the failure in your Amazon CloudWatch Logs logs. Common error scenarios include:

Malformed hash

The password hash doesn't match the expected format for the specified algorithm. Verify that the hash format is correct and matches one of the formats listed in Supported password hashing algorithms.

Parameter out of bounds

The password hash contains parameter values (such as cost factor or iterations) that exceed the maximum allowed values. For users with incompatible hashes, choose one of the following options:

- Import those users without a password hash by leaving the `password_hash` field empty. Amazon Cognito sets them to the `RESET_REQUIRED` state and they must reset their passwords on first sign-in.
- Use a user migration Lambda trigger to migrate those users at sign-in time instead of through CSV import.
- Before decommissioning your source system, re-hash passwords with compatible parameters when users sign in to your existing system.

Algorithm mismatch

The password hash was created with a different algorithm than the one specified in the import job. All hashes in a single import job must use the same algorithm. If you have users with different algorithms, create separate import jobs for each algorithm.

Missing password_hash column

You specified a password hashing algorithm when creating the job, but your CSV file doesn't include the `password_hash` column. Add the column to your CSV file or create a new job without specifying an algorithm.

When a password hash import fails for a user, Amazon Cognito doesn't create the user. Other users in the import job continue to be processed. Review your Amazon CloudWatch Logs logs to identify which users failed and the reason for each failure.

Security considerations for password hash import

Amazon Cognito applies additional security measures to imported password hashes:

- **Double hashing:** Amazon Cognito re-hashes all imported password hashes with an additional layer of cryptographic protection before storage, regardless of the original algorithm's strength.
- **Automatic migration:** After successful first authentication, Amazon Cognito migrates the user's credentials to the native Secure Remote Password (SRP) protocol used by Amazon Cognito. This ensures all users eventually use the native high-security authentication mechanism.
- **Data cleanup:** After the import job completes, Amazon Cognito removes the uploaded CSV file containing password hashes from temporary storage.
- **Password policy not enforced:** Amazon Cognito doesn't enforce the password policy that you configured for the user pool when you import users with password hashes. If a password doesn't meet the password policy that you configured, Amazon Cognito still imports the user. To enforce password strength and reject passwords that don't meet the policy, validate password strength in your source system before you generate hashes for import.

> **Important**
>
> Treat your CSV file containing password hashes as sensitive data. Protect it in transit and delete it securely after the import completes.

Working with user attributes

Attributes are pieces of information that help you identify individual users, such as name, email address, and phone number. A new user pool has a set of default *standard attributes*. You can also

add custom attributes to your user pool definition in the AWS Management Console. This topic describes those attributes in detail and gives you tips on how to set up your user pool.

Don't store all information about your users in attributes. For example, keep user data that changes frequently, such as usage statistics or game scores, in a separate data store, such as Amazon Cognito Sync or Amazon DynamoDB.

Sanitize the inputs for user-attribute string values before you submit them to your user pool. One method to analyze proposed user attribute values is with a Lambda trigger like pre sign-up.

> **Note**
>
> Some documentation and standards refer to attributes as *members*.

Topics

- Standard attributes
- Username and preferred username
- Customizing sign-in attributes
- Custom attributes
- Attribute permissions and scopes

Standard attributes

Amazon Cognito assigns all users a set of standard attributes based on the OpenID Connect specification. By default, standard and custom attribute values can be any string with a length of up to 2048 characters, but some attribute values have format restrictions.

The standard attributes are:

- `name`
- `family_name`
- `given_name`
- `middle_name`
- `nickname`
- `preferred_username`

- profile
- picture
- website
- gender
- birthdate
- zoneinfo
- locale
- updated_at
- address
- email
- phone_number
- sub

Except for sub, standard attributes are optional by default for all users. To make an attribute required, during the user pool creation process, select the **Required** check box next to the attribute. Amazon Cognito assigns a unique user identifier value to each user's sub attribute. Only the **email** and **phone_number** attributes can be verified.

Standard attributes have predefined properties that you can view in the SchemaAttributes parameter of a DescribeUserPool API response. You can set custom values for these attribute properties, like data type, mutability, and length constraints. To modify standard attribute properties, set their custom values in the CreateUserPool Schema parameter. The schema is also where you set required attributes. You can't modify the properties of standard attributes when you create user pools in the Amazon Cognito console.

> **Note**
>
> When you mark a standard attribute as **Required**, a user can't register unless they provide a value for the attribute. To create users and not give values for required attributes, administrators can use the AdminCreateUser API. After you create a user pool, you can't switch an attribute between required and not required.

Standard attribute details and format restrictions

birthdate

Value must be a valid 10 character date in the format YYYY-MM-DD.

email

Users and administrators can verify email address values.

An administrator with proper AWS account permissions can change the user's email address and also mark it as verified. Mark an email address as verified with the AdminUpdateUserAttributes API or the admin-update-user-attributes AWS Command Line Interface (AWS CLI) command. With this command, the administrator can change the `email_verified` attribute to `true`. You can also edit a user in the **Users** menu of the Amazon Cognito console to mark an email address as verified.

Value must be a valid email address string following the standard email format with @ symbol and domain, up to 2048 characters in length.

phone_number

A user must provide a phone number if SMS multi-factor authentication (MFA) is active. For more information, see Adding MFA to a user pool.

Users and administrators can verify phone number values.

An administrator with proper AWS account permissions can change the user's phone number and also mark it as verified. Mark a phone number as verified with the AdminUpdateUserAttributes API or the admin-update-user-attributes AWS CLI command. With this command, the administrator can change the phone_number_verified attribute to `true`. You can also edit a user in the **Users** menu of the Amazon Cognito console to mark a phone number as verified.

> **Important**
>
> Phone numbers must follow these format rules: A phone number must start with a plus (**+**) sign, followed immediately by the country code. A phone number can only contain the **+** sign and digits. Remove any other characters from a phone number, such as parentheses, spaces, or dashes (**-**) before you submit the value to the service.

For example, a phone number based in the United States must follow this format: **+14325551212**.

preferred_username

You can select `preferred_username` as required or as an alias, but not both. If the `preferred_username` is an alias, you can make a request to the UpdateUserAttributes API operation and add the attribute value after you confirm the user.

sub

Index and search your users based on the `sub` attribute. The `sub` attribute is a unique user identifier within each user pool. Users can change attributes like `phone_number` and `email`. The `sub` attribute has a fixed value. For more information about finding users, see Managing and searching for user accounts.

Important

Amazon Cognito generates `sub` in an Amazon Cognito-specific format that doesn't conform to a specific UUID format, including RFC UUID. You shouldn't strictly validate the format of `sub`.

View required attributes

Use the following procedure to view required attributes for a given user pool.

Note

You can't change required attributes after you create a user pool.

To view required attributes

1. Go to Amazon Cognito in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.

4. Choose the **Sign-up** menu.
5. In the **Required attributes** section, view the required attributes of your user pool.

Username and preferred username

The `username` value is a separate attribute and not the same as the `name` attribute. Each user has a `username` attribute. Amazon Cognito automatically generates a username for federated users. You must provide a `username` attribute to create a local user in the Amazon Cognito directory. After you create a user, you can't change the value of the `username` attribute.

Developers can use the `preferred_username` attribute to give users usernames that they can change. For more information, see Customizing sign-in attributes.

If your application doesn't require a username, you don't need to ask users to provide one. Your app can create a unique username for users in the background. This can be useful if you want users to register and sign in with an email address and password. For more information, see Customizing sign-in attributes.

The `username` must be unique within a user pool. A `username` can be reused, but only after you delete it and it is no longer in use. For information about the string constraints to the `username` attributes, see the *username* property of a SignUp API request.

Customizing sign-in attributes

When you create a user pool, you can set up *username attributes* if you want your users to be able to sign up and sign in with an email address or phone number as their username. Alternatively, you can set up *alias attributes* to give your users the option: they can include multiple attributes when they sign up, and then sign in with a username, preferred username, email address, or phone number.

> **Important**
>
> After you create a user pool, you can't change this setting.

How to choose between alias attributes and username attributes

Your requirement	Alias attributes	Username attributes
Users have multiple sign-in attributes	Yes[1]	No[2]
Users must verify email address or phone number before they can sign in with it	Yes	No
Sign up users with duplicate email addresses or phone numbers and prevent `UsernameExistsException` errors[3]	Yes	No
Can assign the same email address or phone number attribute value to more than one user	Yes[4]	No

[1] Available sign-in attributes are username, email address, phone number, and preferred username.

[2] Can sign in with email address or phone number.

[3] Your user pool doesn't generate `UsernameExistsException` errors when users register with potentially-duplicate email addresses or phone numbers, but no username. This behavior is independent of **Prevent username existence errors**, which applies to sign-in, but not sign-up, operations.

[4] Only the last user who has verified the attribute can sign in with it.

Option 1: Multiple sign-in attributes (alias attributes)

An attribute is an *alias* when users have a username but can also sign in with that attribute. Set up aliases when you want to allow your users to choose between their username and other attribute values in the username field of your sign-in form. The `username` attribute is a fixed value that users can't change. If you mark an attribute as an alias, users can sign in with that attribute in

place of the username. You can mark the email address, phone number, and preferred username attributes as aliases. For example, if you select email address and phone number as aliases for a user pool, users in that user pool can sign in with their username, email address, or phone number, along with their password.

To choose alias attributes, select **User name** and at least one additional sign-in option when you create your user pool.

> **Note**
>
> When you configure your user pool to be case insensitive, a user can use either lowercase or uppercase letters to sign up or sign in with their alias. For more information, see CreateUserPool in the *Amazon Cognito user pools API Reference*.

If you select email address as an alias, Amazon Cognito doesn't accept a username that matches a valid email address format. Similarly, if you select phone number as an alias, Amazon Cognito doesn't accept a username for that user pool that matches a valid phone number format.

> **Note**
>
> Alias values must be unique in a user pool. If you configure an alias for an email address or phone number, the value that you provide can be in a verified state in only one account. During sign-up, if your user provides an email address or phone number as an alias value and another user has already used that alias value, registration succeeds. However, when a user tries to confirm the account with this email (or phone number) and enters the valid code, Amazon Cognito returns an `AliasExistsException` error. The error indicates to the user that an account with this email address (or phone number) already exists. At this point, the user can abandon their attempt to create the new account and instead try to reset the password for the old account. If the user continues to create the new account, your app must call the `ConfirmSignUp` API with the `forceAliasCreation` option. `ConfirmSignUp` with `forceAliasCreation` moves the alias from the previous account to the newly created account, and marks the attribute unverified in the previous account. Aliases can also transfer between accounts when a user verifies an attribute with `VerifyUserAttribute`. If a user verifies an email address or phone number that is already a verified alias on another account, Amazon Cognito transfers the alias to their

account and marks the attribute as unverified on the original account. This transfer doesn't raise an `AliasExistsException`.

Phone numbers and email addresses only become active aliases for a user after your user verifies the phone numbers and email addresses. We recommend that you choose automatic verification of email addresses and phone numbers if you use them as aliases.

Choose alias attributes to prevent `UsernameExistsException` errors for email address and phone number attributes when your users sign up.

Activate the `preferred_username` attribute so that your user can change the username that they use to sign in while their `username` attribute value doesn't change. If you want to set up this user experience, submit the new `username` value as a `preferred_username` and choose `preferred_username` as an alias. Then users can sign in with the new value that they entered. If you select `preferred_username` as an alias, your user can provide the value only when they confirm an account. They can't provide the value during registration.

When the user signs up with a username, you can choose if they can sign in with one or more of the following aliases.

- Verified email address
- Verified phone number
- Preferred username

After the user signs up, they can change these aliases.

Important

When your user pool supports sign-in with aliases and you want to authorize or look up a user, don't identify your user by any of their sign-in attributes. The fixed-value user identifier `sub` is the only consistent indicator of your user's identity.

Include the following steps when you create the user pool so that users can sign in with an alias.

Phone number or email address (console)

You must set email address and phone number as alias attributes when you create a user pool.

To create a user pool with username aliases in the Amazon Cognito console

1. Go to Amazon Cognito in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Create a new user pool with the **Get started** or **Create user pool** button.
3. Choose application settings in **Define your application**.
4. In **Configure options** under **Options for sign-in identifiers**, select the checkbox next to **Username** and at least one of the other options, **Email** and **Phone number**.
5. Choose your alias attributes as **Required attributes for sign-up**. In the managed login sign-up form, Amazon Cognito prompts new users to provide values for required attributes.
6. Under **Add a return URL**, set up an application callback URL for redirect after managed login sign-in.
7. Choose **Create**.

Phone number or email address (API/SDK)

Create a new user pool with the CreateUserPool API operation. Configure the `AliasAttributes` parameter as shown. You can remove the `email` entry if you only want phone-number aliases, or remove the `phone_number` entry if you only want email-address aliases.

```
"AliasAttributes": [
   "email",
   "phone_number"
],
```

Preferred username (API/SDK)

The Amazon Cognito console creates user pools without `preferred_username` as an alias. To create user pools with a `preferred_username` alias, set up user pools with CreateUserPool API requests in an AWS SDK. To support the creation of preferred username attributes at sign-up, set `preferred_username` as a required attribute. In the managed login sign-up form, Amazon Cognito prompts new users to provide values for required attributes. You *can* set `preferred_username` as a required attribute in the Amazon Cognito console, but this doesn't make it available as an alias.

Configure as an alias

Configure `preferred_username` as an alias in the `AliasAttributes` parameter of a `CreateUserPool` request as shown. Remove any values that you don't want as alias attributes from the list.

```
"AliasAttributes": [
   "email",
   "phone_number",
   "preferred_username"
],
```

Configure as required

In the managed login sign-up form, Amazon Cognito prompts new users to provide values for required attributes. Configure `preferred_username` as required in the `SchemaAttributes` parameter of a CreateUserPool request.

To set preferred username as a required attribute, configure it as shown. The following example modifies the default schema of `preferred_username` to set it as required. Other schema parameters like `AttributeDataType` (defaults to `string`) and `StringAttributeConstraints` (defaults to 1-99 characters in length) assume default values.

```
"Schema": [
   {
      "Name": "preferred_username",
      "Required": true
   }
]
```

Option 2: Email address or phone number as a sign-in attribute (username attributes)

When the user signs up with an email address or phone number as their username, you can choose if they can sign up with only email addresses, only phone numbers, or either one.

To choose username attributes, don't select **Username** as a sign-in option when you create your user pool.

The email address or phone number must be unique, and it must not already be in use by another user. It doesn't have to be verified. After the user has signed up with an email address or phone

number, the user can't create a new account with the same email address or phone number. The user can only reuse the existing account and reset the account password, if needed. However, the user can change the email address or phone number to a new email address or phone number. If the email address or phone number isn't already in use, it becomes the new username.

When you select both email address and phone number as username attributes, users can sign in with one or the other, even if they provide values for both attributes. The sign-in username is based on the value that you pass in the `Username` parameter of SignUp.

> **Note**
>
> If a user signs up with an email address as their username, they can change the username to another email address, but they can't change it to a phone number. If they sign up with a phone number, they can change the username to another phone number, but they can't change it to an email address.

Use the following steps during the user pool creation process to set up sign-up and sign-in with email address or phone number.

Username attributes (console)

The following procedure creates a user pool with email address or phone number username attributes. The difference in the process for username attributes in the Amazon Cognito console is that you don't also set **Username** as a sign-in attribute.

To create a user pool with username attributes in the Amazon Cognito console

1. Go to Amazon Cognito in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Create a new user pool with the **Get started** or **Create user pool** button.
3. Choose application settings in **Define your application**.
4. In **Configure options** under **Options for sign-in identifiers**, select your username attributes: **Email**, **Phone number**, or both. Leave **Username** unchecked.
5. As a best practice, select your username attributes as **Required attributes for sign-up**. In the managed login sign-up form, Amazon Cognito prompts new users to provide values for required attributes. If you don't set your username attributes as required, Amazon Cognito doesn't prompt new users to provide values for them. In that scenario, you must configure

your application to collect and submit email addresses or phone numbers for each user before they can sign in.

6. Under **Add a return URL**, set up an application callback URL for redirect after managed login sign-in.
7. Choose **Create**.

Username attributes (API/SDK)

In a CreateUserPool request, configure the `UsernameAttributes` parameter as shown. To allow sign-in only with email-address usernames, specify `email` alone in this list. To allow sign-in only with phone-number usernames, specify `phone_number` alone. This parameter overrides username as a sign-in option.

```
"UsernameAttributes": [
   "email",
   "phone_number"
],
```

The following example configures a user pool where each user signs in with only their email address as the username. As a best practice, also set `email` as a required sign-up attribute so that new users provide their email address when they sign up through managed login:

```
"UsernameAttributes": [
   "email"
],
```

When you configure username attributes, your can make SignUp API requests that pass an email address or phone number in the `username` parameter. The following is the behavior of the code`SignUp` API operation with username attributes.

- If the `username` string is in valid email address format, for example `user@example.com`, the user pool automatically populates the `email` attribute of the user with the `username` value.
- If the `username` string is in valid phone number format, for example `+12065551212`, the user pool automatically populates the `phone_number` attribute of the user with the `username` value.
- If the `username` string format isn't in email address or phone number format, the `SignUp` API returns an exception.

- If the `username` string contains an email address or phone number that is already in use, the `SignUp` API returns an exception.
- The `SignUp` API populates the `username` attribute with a UUID for your user. This UUID has the same value as the `sub` claim in the user identity token.

You can use an email address or phone number in place of the username in all APIs except the `ListUsers` operation. This includes post-sign-up operations such as `ConfirmSignUp` and `ResendConfirmationCode`. Although `SignUp` sets the `username` attribute to the UUID, you can still pass the user's email address or phone number as the `Username` value in the `SignUp` request. In `ListUsers` API requests, you can specify a `Filter` of `email` or `phone_number`. To filter by `username`, supply the UUID username. You can't filter by the email address or phone number in a `username` filter.

Custom attributes

You can add up to 50 custom attributes to your user pool. You can specify a minimum and/or maximum length for custom attributes. However, the maximum length for any custom attribute can be no more than 2048 characters. The name of a custom attribute must match the regular expression pattern that's described in the `Name` parameter of `SchemaAttributeType`.

> **Custom attributes are permanent**
>
> You can't remove or change a custom attribute after you add it to your user pool. You can't delete or rename the attribute. You also can't change its data type, mutability, or length constraints. Plan your custom attribute schema before you add attributes to a production user pool.

Each custom attribute has the following characteristics:

- You can define it as a string, number, boolean, or `DateTime` object. Amazon Cognito writes custom attribute values to the ID token only as strings.

> **Note**
>
> In the Amazon Cognito console, you can add custom attributes only of the string and number data types. Additional options like boolean and `DateTime` attribute data

types are only available in the `SchemaAttributes` property of CreateUserPool and UpdateUserPool API requests.

> **Note**
>
> Regardless of the data type that you assign, Amazon Cognito stores and transmits every custom attribute value as a string. When you set a value in an API or SDK request — for example AdminUpdateUserAttributes, UpdateUserAttributes, or SignUp — you must pass the value as a string, even for a `Number`, `Boolean`, or `DateTime` attribute. Pass `"42"` rather than `42`, and `"true"` rather than `true`. A native number or boolean is rejected before the value is stored; depending on your SDK, this surfaces as a client-side type-validation error or a service `InvalidParameterException`. Your application is responsible for parsing the string back to its intended type when it reads the attribute.

- You can't require that users provide a value for the attribute.
- You can't remove or change it after you add it to the user pool.
- The character length of the attribute name is within the limit that Amazon Cognito accepts. For more information, see Quotas in Amazon Cognito.
- It can be *mutable* or *immutable*. You can only write a value to an immutable attribute when you create a user. You can change the value of a mutable attribute if your app client has write permission to the attribute. See Attribute permissions and scopes for more information.

> **Note**
>
> In your code, and in rules settings for Using role-based access control, custom attributes require the `custom:` prefix to distinguish them from standard attributes.

You can also add *developer attributes* when you create user pools, in the `SchemaAttributes` property of CreateUserPool. Developer attributes have a `dev:` prefix. You can only modify a user's developer attributes with AWS credentials. Developer attributes are a legacy feature that Amazon Cognito replaced with app client read-write permissions.

Use the following procedure to create a new custom attribute.

To add a custom attribute using the console

1. Go to Amazon Cognito in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Sign-up** menu, and in the **Custom attributes** section, choose **Add custom attributes**.
5. On the **Add custom attributes** page, provide the following details about the new attribute:
   - Enter a **Name**.
   - Select a **Type** of either **String** or **Number**.
   - Enter a **Min** string length or number value.
   - Enter a **Max** string length or number value.
   - Select **Mutable** if you want to give users permission to change the value of a custom attribute after they set the initial value.
6. Choose **Save changes**.

Attribute permissions and scopes

For each app client, you can set read and write permissions for each user attribute. This way, you can control the access that any app has to read and modify each attribute that you store for your users. For example, you might have a custom attribute that indicates whether a user is a paying customer or not. Your apps might be able to see this attribute but not change it directly. Instead, you would update this attribute using an administrative tool or a background process. You can set permissions for user attributes from the Amazon Cognito console, the Amazon Cognito API, or the AWS CLI. By default, any new custom attributes aren't available until you set read and write permissions for them. By default, when you create a new app client, you grant your app read and write permissions for all standard and custom attributes. To limit your app to only the amount of information that it requires, assign specific permissions to attributes in your app client configuration.

As a best practice, specify attribute read and write permissions when you create an app client. Grant your app client access to the minimum set of user attributes that you need for the operation of your application.

Important

When you don't specify `WriteAttributes` for an app client, the app client has write access to all standard and custom attributes. Users can change the value of these attributes for themselves with `UpdateUserAttributes`.
Your application or identity pool might make authorization decisions based on an attribute. In that case, remove the attribute from the `WriteAttributes` of every app client in your user pool. User attributes are shared by all app clients in a user pool. A value that a user writes through one app client appears in the tokens that all app clients issue.

Note

DescribeUserPoolClient only returns values for `ReadAttributes` and `WriteAttributes` when you configure app client permissions other than the default.

To update attribute permissions (AWS Management Console)

1. Go to Amazon Cognito in the AWS Management Console. If the console prompts you, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **App clients** menu and choose an app client from the list.
5. In the **Attribute permissions** tab, choose **Edit**.
6. On the **Edit attribute read and write permissions** page, configure your read and write permissions, and then choose **Save changes**.

Repeat these steps for each app client that uses the custom attribute. To make an attribute read-only in your apps, remove write permission for that attribute from every app client in your user pool.

For each app client, you can mark attributes as readable or writeable. This applies to both standard and custom attributes. Your app can retrieve the value of attributes that you mark as readable, and can set or modify the value of attributes that you mark as writeable. If your app tries to set a value for an attribute that it isn't authorized to write, Amazon Cognito returns

`NotAuthorizedException`. GetUser requests include an access token with an app client claim; Amazon Cognito only returns values for attributes that your app client can read. Your user's ID token from an app only contains claims that correspond to the readable attributes. All app clients can write user pool required attributes. You can only set the value of an attribute in an Amazon Cognito user pools API request when you also provide a value for any required attributes that don't yet have a value.

Custom attributes have distinct features for read and write permissions. You can create them as mutable or immutable for the user pool, and you can set them as read or write attributes for any app client.

An immutable custom attribute can be updated once, during user creation. You can populate an immutable attribute with the following methods.

- `SignUp`: A user signs up with an app client that has write access to an immutable custom attribute. They provide a value for that attribute.
- Sign-in with a third-party IdP: A user signs in to an app client that has write access to an immutable custom attribute. Your user pool configuration for their IdP has a rule to map a provided claim to an immutable attribute. This is possible but not practical, because the user will only be able to sign in one time. On sign-in attempts after their first, Amazon Cognito rejects the attempt because of the mapping rule to a now-unwriteable attribute.
- `AdminCreateUser`: You provide a value for an immutable attribute.

Attribute permissions with scopes

In user pools that you configure with an AWS SDK or CDK, the REST API, or the AWS CLI, you can configure app client read or write access with the OIDC scope `oidc:profile`. The `oidc:profile` grants read or write access to the following standard attributes:

- `name`
- `family_name`
- `given_name`
- `middle_name`
- `nickname`
- `preferred_username`
- `profile`

- `picture`
- `website`
- `gender`
- `birthdate`
- `zoneinfo`
- `locale`

This list is the OIDC standard attributes minus `email`, `phone_number`, `sub`, and `address`, as defined in section 2.4 of the OIDC specification. For information about the scopes that you can assign to your app clients, see Scopes, M2M, and resource servers.

To configure your app client to write to the attributes under the `oidc:profile` scope, set the value of WriteAttributes to `oidc:profile`, plus any other attributes that you want to permit your application to modify, in a CreateUserPoolClient or UpdateUserPoolClient API request. Similarly, to grant read access to these attributes, add `oidc:profile` to the value of ReadAttributes.

You can change attribute permissions and scopes after you have created your user pool.

Updating user pool and app client configuration

When you want to change a setting in a user pool or app client, you can apply the update in the Amazon Cognito console with a few clicks. You navigate through the feature-based tabs in your user pool settings and update fields as described in other areas of this guide.

Many organizations manage their resources programmatically in AWS CloudFormation, applications built on the AWS SDKs or CDK, and other automation software. When this is your resource-management model, you must take extra care when you stage changes to your resources.

The API operations UpdateUserPool and UpdateUserPoolClient make updates to an existing user pool or app client. Each comes with a warning in the API Reference: *If you don't provide a value for an attribute, Amazon Cognito sets it to its default value.* When you submit an update request with just one parameter, Amazon Cognito sets that parameter to the value of your choosing and sets all others to a default value. This can reset configurations including your attribute schema, your Lambda triggers, and your email and SMS message configuration.

Additionally, some settings are locked in after you create your user pool or app client, and you can't change them unless you create a new resource.

Topics

- Settings you can't change
- SMS configuration
- Updating a user pool with an AWS SDK, AWS CDK, or REST API

Settings you can't change

You can't change some settings after you create a user pool. If you want to change the following settings, you must create a new user pool or app client.

> **Note**
>
> Previously, you couldn't change the name of a user pool. This has changed. You can now assign new friendly names to your user pools.

User pool ID

API parameter name: Id/UserPoolId

The user pool ID, for example `us-east-1_EXAMPLE`, is automatically generated by Amazon Cognito and can't be changed.

Amazon Cognito user pool sign-in options

API parameter names: AliasAttributes and UsernameAttributes

The attributes that your users can pass as a user name when they sign in. When you create a user pool, you can choose to allow sign-in with user name, email address, phone number, or a preferred user name. To change user pool sign-in options, create a new user pool.

Make user name case sensitive

API parameter name: UsernameConfiguration

When you create a user name that matches another user name except for the letter case, Amazon Cognito can treat them as either the same user or as unique users. For more information, see User pool case sensitivity. To change case sensitivity, create a new user pool.

Client secret

API parameter name: GenerateSecret

When you create an app client, you can generate a client secret so that only trusted sources can make requests to your user pool. For more information, see Application-specific settings with app clients. To change a client secret, create a new app client in the same user pool.

Required attributes

API parameter name: Schema

The attributes that your users must provide values for when they sign up, or when you create them. For more information, see Working with user attributes. To change required attributes, create a new user pool.

Custom attributes (deletion)

API parameter name: Schema

Attributes with custom names. You can change the value of a user's custom attribute, but you can't delete a custom attribute from your user pool. For more information, see Working with user attributes. If you reach the maximum number of custom attributes and you want to modify the list, create a new user pool.

SMS configuration

After you activate SMS messages in your user pool, you can't deactivate them.

- If you choose to configure SMS messages when you create a user pool, you can't deactivate SMS after you complete setup.
- You can activate SMS messages in a user pool that you created, but after that you can't deactivate SMS.
- Amazon Cognito can use SMS messages for user account invitation and recovery, attribute verification, and multi-factor authentication (MFA). After you activate SMS messages, you can turn SMS messages on or off for these functions at any time.
- SMS message configuration includes an IAM role that you delegate to Amazon Cognito to send messages with Amazon SNS. You can change the assigned role at any time.

Updating a user pool with an AWS SDK, AWS CDK, or REST API

In the Amazon Cognito console, you can change your user pool settings one parameter at a time. For example, to add a Lambda trigger, you choose **Add Lambda trigger** and choose the function

and trigger type. The Amazon Cognito user pools API is structured in a way that update operations for user pools and app clients require the full set of parameters for the user pool. However, the console transparently automates this update operation with your other user pool settings.

You might find at times that a change elsewhere in your AWS account can cause updates to generate an error when they aren't related to the setting you want to change. A deleted Amazon SES identity or a change in an IAM permission for AWS WAF, for example. If one of the current parameters is no longer valid, you can't update your settings until you fix it. When you encounter such an error, examine the error response and validate the setting that it mentions.

The AWS Cloud Development Kit (AWS CDK), Amazon Cognito user pools REST API and AWS SDKs are tools for automation and programmatic configuration of Amazon Cognito resources. Requests with these tools must also, like the Amazon Cognito console, update a setting with a full resource configuration in the request body. At a high level, you must perform the following process.

1. Capture the output from an operation that describes the configuration of your existing resource .
2. Modify the output with your settings change.
3. Send the modified configuration in an operation that updates your resource.

The following procedure updates your configuration with the UpdateUserPool API operation. The same approach, with different input fields, applies to UpdateUserPoolClient.

> **Important**
>
> If you don't provide values for existing parameters, Amazon Cognito sets them to default values. For example, when you have existing `LambdaConfig` and you submit an `UpdateUserPool` with an empty `LambdaConfig`, you delete the assignment of all Lambda functions to user pool triggers. Plan accordingly when you want to automate changes to your user pool configuration.

1. Capture the existing state of your user pool with DescribeUserPool.
2. Format the output of `DescribeUserPool` to match the request parameters of `UpdateUserPool`. Remove the following top-level fields and their child objects from the output JSON.

   - `Arn`

- `CreationDate`
- `CustomDomain`
  - Update this field with the UpdateUserPoolDomain API operation.
- `Domain`
  - Update this field with the UpdateUserPoolDomain API operation.
- `EmailConfigurationFailure`
- `EstimatedNumberOfUsers`
- `Id`
- `LastModifiedDate`
- `Name`
- `SchemaAttributes`
- `SmsConfigurationFailure`
- `Status`

3. Confirm that the resulting JSON matches the request parameters of `UpdateUserPool`.
4. Modify any parameters that you want to change in the resulting JSON.
5. Submit an `UpdateUserPool` API request with your modified JSON as the request input.

You can also use this modified `DescribeUserPool` output in the `--cli-input-json` parameter of `update-user-pool` in the AWS CLI.

Alternately, run the following AWS CLI command to generate JSON with blank values for the accepted input fields for `update-user-pool`. You can then populate these fields with the existing values from your user pool.

```
aws cognito-idp update-user-pool --generate-cli-skeleton --output json
```

Run the following command to generate the same JSON object for an app client.

```
aws cognito-idp update-user-pool-client --generate-cli-skeleton --output json
```

Customizing user pools

Cognito user pools expose extension points so that you can shape the sign-up and sign-in experience to your application. Use Lambda triggers to run your own code at key moments in the user lifecycle, and configure the messages and analytics that your user pool sends.

Topics

- Customizing user pool workflows with Lambda triggers
- Messaging and analytics

Customizing user pool workflows with Lambda triggers

Amazon Cognito works with AWS Lambda functions to modify the authentication behavior of your user pool. You can configure your user pool to automatically invoke Lambda functions before their first sign-up, after they complete authentication, and at several stages in between. Your functions can modify the default behavior of your authentication flow, make API requests to modify your user pool or other AWS resources, and communicate with external systems. The code in your Lambda functions is your own. Amazon Cognito sends event data to your function, waits for the function to process the data, and in most cases anticipates a response event that reflects any changes you want to make to the session.

Within the system of request and response events, you can introduce your own authentication challenges, migrate users between your user pool and another identity store, customize messages, and modify JSON web tokens (JWTs).

Lambda triggers can customize the response that Amazon Cognito delivers to your user after they initiate an action in your user pool. For example, you can prevent sign-in by a user who would otherwise succeed. They can also perform runtime operations against your AWS environment, external APIs, databases, or identity stores. The migrate user trigger, for example, can combine external action with a change in Amazon Cognito: you can look up user information in an external directory, then set attributes on a new user based on that external information.

When you have a Lambda trigger assigned to your user pool, Amazon Cognito interrupts its default flow to request information from your function. Amazon Cognito generates a JSON *event* and passes it to your function. The event contains information about your user's request to create a user account, sign in, reset a password, or update an attribute. Your function then has an

opportunity to take action, or to send the event back unmodified. An event returned unmodified notifies your user pool to proceed with the default action for the event. For example, your pre sign-up trigger can automatically confirm users for the `PreSignUp_SignUp` trigger source, but return the event unchanged for external and administrator-created users.

The following table summarizes some of the ways you can use Lambda triggers to customize user pool operations:

User Pool Flow	Operation	Description
Custom Authentication Flow	**Define Auth Challenge**	Determines the next challenge in a custom auth flow
	Create Auth Challenge	Creates a challenge in a custom auth flow
	Verify Auth Challenge Response	Determines if a response is correct in a custom auth flow
Authentication Events	**the section called “Pre authentication”**	Custom validation to accept or deny the sign-in request
	the section called “Post authentication”	Logs events for custom analytics
	the section called “Pre token generation”	Augments or suppresses token claims
Federation	**the section called “Inbound federation”**	Transforms federated user attributes before user creation or update in Amazon Cognito user pools
Sign-Up	**the section called “Pre sign-up”**	Performs custom validatio n that accepts or denies the sign-up request

User Pool Flow	Operation	Description
	the section called “Post confirmation”	Adds custom welcome messages or event logging for custom analytics
	the section called “Migrate user”	Migrates a user from an existing user directory to user pools
Messages	the section called “Custom message”	Performs advanced customization and localization of messages
Token Creation	the section called “Pre token generation”	Adds or removes attributes in ID and access tokens
Email and SMS third-party providers	the section called “Custom senders”	Uses a third-party provider to send SMS and email messages

Topics

- Things to know about Lambda triggers
- Add a user pool Lambda trigger
- User pool Lambda trigger event
- User pool Lambda trigger common parameters
- Client metadata
- Connecting API operations to Lambda triggers
- Connecting Lambda triggers to user pool functional operations
- Pre sign-up Lambda trigger
- Post confirmation Lambda trigger
- Pre authentication Lambda trigger
- Post authentication Lambda trigger
- Inbound federation Lambda trigger

- Custom authentication challenge Lambda triggers
- Pre token generation Lambda trigger
- Migrate user Lambda trigger
- Custom message Lambda trigger
- Custom sender Lambda triggers

Things to know about Lambda triggers

When you are preparing your user pools for Lambda functions, consider the following:

- The events that Amazon Cognito sends to your Lambda triggers might change with new features. The positions of response and request elements in the JSON hierarchy might change, or element names might be added. In your Lambda function, you can expect to receive the input-element key-value pairs described in this guide, but stricter input validation can cause your functions to fail.
- You can choose one of multiple versions of the events that Amazon Cognito sends to some triggers. Some versions might require you to accept a change to your Amazon Cognito pricing. For more information about pricing, see Amazon Cognito Pricing. To customize access tokens in a Pre token generation Lambda trigger, you must configure your user pool with a feature plan other than *Lite* and update your Lambda trigger configuration to use event version 2.
- Except for Custom sender Lambda triggers, Amazon Cognito invokes Lambda functions synchronously. When Amazon Cognito calls your Lambda function, it must respond within 5 seconds. If it doesn't and if the call can be retried, Amazon Cognito may retry the call. If all retry attempts fail, the function times out. You can't change this five-second timeout value. For more information, see Lambda programming model in the AWS Lambda Developer Guide.

  Amazon Cognito doesn't retry function calls that return an Invoke error with an HTTP status code of 500-599. These codes indicate a configuration issue that leaves Lambda unable to launch the function. For more information, see Error handling and automatic retries in AWS Lambda.
- You can't declare a function version in your Lambda trigger configuration. Amazon Cognito user pools invoke the latest version of your function by default. However, you can associate a function version with an alias and set your trigger `LambdaArn` to the alias ARN in a CreateUserPool or UpdateUserPool API request. This option isn't available in the AWS Management Console. For more information about aliases, see Lambda function aliases in the *AWS Lambda Developer Guide*.

- If you delete a Lambda trigger, you must update the corresponding trigger in the user pool. For example, if you delete the post authentication trigger, you must set the **Post authentication** trigger in the corresponding user pool to **none**.
- If your Lambda function doesn't return the request and response parameters to Amazon Cognito, or returns an error, the authentication event doesn't succeed. You can return an error in your function to prevent a user's sign-up, authentication, token generation, or any other stage of their authentication flow that invokes Lambda trigger.

  Managed login returns errors that Lambda triggers generate as error text above the sign-in prompt. The Amazon Cognito user pools API returns trigger errors in the format *`[trigger]`* `failed with error` *`[error text from response]`*. As a best practice, only generate errors in your Lambda functions that you want your users to see. Use output methods like `print()` to log any sensitive or debugging information to CloudWatch Logs. For an example, see Pre sign-up example: Deny sign-up if user name has fewer than five characters.
- You can add a Lambda function in another AWS account as a trigger for your user pool. You must add cross-account triggers with the CreateUserPool and UpdateUserPool API operations, or their equivalents in CloudFormation and the AWS CLI. You can't add cross-account functions in the AWS Management Console.
- When you add a Lambda trigger in the Amazon Cognito console, Amazon Cognito adds a resource-based policy to your function that permits your user pool to invoke the function. When you create a Lambda trigger outside of the Amazon Cognito console, including a cross-account function, you must add permissions to the resource-based policy of the Lambda function. Your added permissions must allow Amazon Cognito to invoke the function on behalf of your user pool. You can add permissions from the Lambda Console or use the Lambda AddPermission API operation.

  Example Lambda Resource-Based Policy

  The following example Lambda resource-based policy grants Amazon Cognito a limited ability to invoke a Lambda function. Amazon Cognito can only invoke the function when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

  JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
```

```
    "Statement": [
        {
            "Sid": "LambdaCognitoIdpTrust",
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-idp.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:eusc-de-
east-1:111122223333:function:MyFunction",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "AWS:SourceArn": "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                }
            }
        }
    ]
}
```

Add a user pool Lambda trigger

To add a user pool Lambda trigger with the console

1. Use the Lambda console to create a Lambda function. For more information on Lambda functions, see the AWS Lambda Developer Guide.
2. Go to the Amazon Cognito console, and then choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Extensions** menu and locate **Lambda triggers**.
5. Choose **Add a Lambda trigger**.
6. Select a Lambda trigger **Category** based on the stage of authentication that you want to customize.
7. Select **Assign Lambda function** and select a function in the same AWS Region as your user pool.

> **Note**
>
> If your AWS Identity and Access Management (IAM) credentials have permission to update the Lambda function, Amazon Cognito adds a Lambda resource-based policy. With this policy, Amazon Cognito can invoke the function that you select. If the signed-in credentials do not have sufficient IAM permissions, you must update the resource-based policy separately. For more information, see the section called “Things to know”.

8. Choose **Save changes**.
9. You can use CloudWatch in the Lambda console to log your Lambda function . For more information, see Accessing CloudWatch Logs for Lambda.

User pool Lambda trigger event

Amazon Cognito passes event information to your Lambda function. The Lambda function returns the same event object back to Amazon Cognito with any changes in the response. If your function returns the input event without modification, Amazon Cognito proceed with default behavior. The following shows the parameters that are common to all Lambda trigger input events. For trigger-specific event syntax, review the event schema on the section of this guide for each trigger.

JSON

```
{
    "version": "string",
    "triggerSource": "string",
    "region": AWSRegion,
    "userPoolId": "string",
    "userName": "string",
    "callerContext":
        {
            "awsSdkVersion": "string",
            "clientId": "string"
        },
    "request":
        {
            "userAttributes": {
                "string": "string",
                ....
            }
```

```
        },
    "response": {}
}
```

User pool Lambda trigger common parameters

version

The version number of your Lambda function.

triggerSource

The name of the event that triggered the Lambda function. For a description of each triggerSource see Connecting Lambda triggers to user pool functional operations.

region

The AWS Region as an `AWSRegion` instance.

userPoolId

The ID of the user pool.

userName

The current user's username.

callerContext

Metadata about the request and the code environment. It contains the fields **awsSdkVersion** and **clientId**.

awsSdkVersion

The version of the AWS SDK that generated the request.

clientId

The ID of the user pool app client.

request

Details of your user's API request. It includes the following fields, and any request parameters that are particular to the trigger. For example, an event that Amazon Cognito sends to a pre-authentication trigger will also contain a `userNotFound` parameter. You can process the value of this parameter to take a custom action when your user tries to sign in with an unregistered username.

userAttributes

One or more key-value pairs of user attribute names and values, for example `"email": "john@example.com"`.

response

This parameter doesn't contain any information in the original request. Your Lambda function must return the entire event to Amazon Cognito, and add any return parameters to the `response`. To see what return parameters your function can include, refer to the documentation for the trigger that you want to use.

Client metadata

You can submit custom parameters to your Lambda trigger functions in API operations and Token endpoint requests. With client metadata, your application can collect additional information about the environment where requests originate. When you pass client metadata to your Lambda functions, they can process the additional data and make use of it in logging or customization of authentication flows. Client metadata is string pairs of your choosing and design in a JSON key-value format.

Client metadata example use cases

- Pass geolocation data at sign-up to the pre sign-up trigger and prevent sign-in from unwanted locations.
- Pass tenant ID data to custom challenge triggers and issue different challenges to customers from different business units.
- Pass a user's token to the pre token generation trigger and generate a log of the principal that an M2M request was made on behalf of. For an example request, see Client credentials with basic authorization.

Here is an example of passing client metadata to the pre sign-up trigger.

SignUp request

The following is an example SignUp request with client metadata that Amazon Cognito passes to a pre sign-up trigger.

```
POST HTTP/1.1
```

```
Host: cognito-idp.us-east-1.amazonaws.com
X-Amz-Date: 20230613T200059Z
Accept-Encoding: gzip, deflate, br
X-Amz-Target: AWSCognitoIdentityProviderService.SignUp
User-Agent: <UserAgentString>
Authorization: AWS4-HMAC-SHA256 Credential=<Credential>, SignedHeaders=<Headers>,
 Signature=<Signature>
Content-Length: <PayloadSizeBytes>

{
    "ClientId": "1example23456789",
    "Username": "mary_major",
    "Password": "<Password>",
    "SecretHash": "<Secret hash>",
    "ClientMetadata": {
        "IpAddress" : "192.0.2.252",
        "GeoLocation" : "Netherlands (Kingdom of the) [NL]"
    }
    "UserAttributes": [
        {
            "Name": "name",
            "Value": "Mary"
        },
        {
            "Name": "email",
            "Value": "mary_major@example.com"
        },
        {
            "Name": "phone_number",
            "Value": "+12065551212"
        }
    ],
}
```

Lambda trigger input event

The request results in the following request body to your pre sign-up function.

```
{
    "callerContext": {
        "awsSdkVersion": "aws-sdk-unknown-unknown",
        "clientId": "1example23456789"
    },
    "region": "us-west-2",
```

```
    "request": {
        "clientMetadata": {
            "GeoLocation": "Netherlands (Kingdom of the) [NL]",
            "IpAddress": "192.0.2.252"
        },
        "userAttributes": {
            "email": "mary_major@example.com",
            "name": "Mary",
            "phone_number": "+12065551212"
        },
        "validationData": null
    },
    "response": {
        "autoConfirmUser": false,
        "autoVerifyEmail": false,
        "autoVerifyPhone": false
    },
    "triggerSource": "PreSignUp_SignUp",
    "userName": "mary_major2",
    "userPoolId": "us-west-2_EXAMPLE",
    "version": "1"
}
```

Client metadata for machine-to-machine (M2M) client credentials

You can pass client metadata in M2M requests. Client metadata is additional information from a user or application environment that can contribute to the outcomes of a Pre token generation Lambda trigger. In authentication operations with a user principal, you can pass client metadata to the pre token generation trigger in the body of AdminRespondToAuthChallenge and RespondToAuthChallenge API requests. Because applications conduct the flow for generation of access tokens for M2M with direct requests to the Token endpoint, they have a different model. In the POST body of token requests for client credentials, pass an `aws_client_metadata` parameter with the client metadata object URL-encoded (`x-www-form-urlencoded`) to string. For an example request, see Client credentials with basic authorization. The following is an example parameter that passes the key-value pairs `{"environment": "dev", "language": "en-US"}`.

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

Temporary user attributes: `validationData`

Some authentication operations also have a `validationData` parameter. Like client metadata, this is an opportunity to pass external information that Amazon Cognito doesn't automatically gather to Lambda triggers. The validation data field is intended to provide your Lambda function with additional user context in sign-up and sign-in operations. SignUp and AdminCreateUser pass `validationData` to the pre sign-up trigger. InitiateAuth and AdminInitiateAuth pass `ClientMetadata` in the API request body as `validationData` in the input event to the pre authentication and migrate user triggers.

To map API operations to the functions that they can pass client metadata to, refer to the trigger source sections that follow.

Connecting API operations to Lambda triggers

The following sections describe the Lambda triggers that Amazon Cognito invokes from the activity in your user pool.

When your app signs in users through the Amazon Cognito user pools API, managed login, or user pool endpoints, Amazon Cognito invokes your Lambda functions based on the session context. For more information about the Amazon Cognito user pools API and user pool endpoints, see Understanding API, OIDC, and managed login pages authentication. The tables in the sections that follow describe events that cause Amazon Cognito to invoke a function, and the `triggerSource` string that Amazon Cognito includes in the request.

Topics

- Lambda triggers in the Amazon Cognito API
- Lambda triggers for Amazon Cognito local users in managed login
- Lambda triggers for federated users

Lambda triggers in the Amazon Cognito API

The following table describes the source strings for the Lambda triggers that Amazon Cognito can invoke when your app creates, signs in, or updates a local user.

Local user trigger sources in the Amazon Cognito API

API operation	Lambda trigger	Trigger source
AdminCreateUser	Pre sign-up	PreSignUp_AdminCreateUser
	Pre token generation	TokenGeneration_NewPasswordChallenge
	Custom message	CustomMessage_AdminCreateUser
	Custom email sender	CustomEmailSender_AdminCreateUser
	Custom SMS sender	CustomSMSSender_AdminCreateUser
SignUp	Pre sign-up	PreSignUp_SignUp
	Custom message	CustomMessage_SignUp
	Custom email sender	CustomEmailSender_SignUp
	Custom SMS sender	CustomSMSSender_SignUp
ConfirmSignUp AdminConfirmSignUp	Post confirmation	PostConfirmation_ConfirmSignUp
InitiateAuth AdminInitiateAuth	Pre authentication	PreAuthentication_Authentication
	Post authentication	PostAuthentication_Authentication
	Define auth challenge	DefineAuthChallenge_Authentication

API operation	Lambda trigger	Trigger source
	Create auth challenge	`CreateAuthChallenge_Authentication`
	Verify auth challenge	`VerifyAuthChallenge_Authentication`

	Pre token generation	`TokenGeneration_Authentication`
		`TokenGeneration_AuthenticateDevice`
		`TokenGeneration_RefreshTokens`

	Migrate user	`UserMigration_Authentication`
	Custom message	`CustomMessage_Authentication`
	Custom email sender	`CustomEmailSender_AccountTakeOverNotification` `CustomEmailSender_Authentication`
	Custom SMS sender	`CustomSMSSender_Authentication`
RespondToAuthChallenge AdminRespondToAuthChallenge	Post authentication	`PostAuthentication_Authentication`
	Define auth challenge	`DefineAuthChallenge_Authentication`

API operation	Lambda trigger	Trigger source
	Create auth challenge	`CreateAuthChallenge_Authentication`
	Verify auth challenge	`VerifyAuthChallenge_Authentication`

	Pre token generation	`TokenGeneration_Authentication`
		`TokenGeneration_AuthenticateDevice`
		`TokenGeneration_RefreshTokens`

	Custom message	`CustomMessage_Authentication`
	Custom email sender	`CustomEmailSender_AccountTakeOverNotification` `CustomEmailSender_Authentication`
	Custom SMS sender	`CustomSMSSender_Authentication`
ForgotPassword	Migrate user	`UserMigration_ForgotPassword`
	Custom message	`CustomMessage_ForgotPassword`
	Custom email sender	`CustomEmailSender_ForgotPassword`

API operation	Lambda trigger	Trigger source
	Custom SMS sender	`CustomSMSSender_ForgotPassword`
ConfirmForgotPassword	Post confirmation	`PostConfirmation_ConfirmForgotPassword`
UpdateUserAttributes AdminUpdateUserAttributes	Custom message	`CustomMessage_UpdateUserAttribute`
	Custom email sender	`CustomEmailSender_UpdateUserAttribute`
	Custom SMS sender	`CustomSMSSender_UpdateUserAttribute`
VerifyUserAttributes	Custom message	`CustomMessage_VerifyUserAttribute`
	Custom email sender	`CustomEmailSender_VerifyUserAttribute`
	Custom SMS sender	`CustomSMSSender_VerifyUserAttribute`
GetTokensFromRefreshToken	Pre token generation	`TokenGeneration_Authentication`

Lambda triggers for Amazon Cognito local users in managed login

The following table describes the source strings for the Lambda triggers that Amazon Cognito can invoke when a local user signs in to your user pool with managed login.

Local user trigger sources in managed login

Managed login URI	Lambda trigger	Trigger source
/signup	Pre sign-up	PreSignUp_SignUp
	Custom message	CustomMessage_SignUp
	Custom email sender	CustomEmailSender_SignUp
	Custom SMS sender	CustomSMSSender_SignUp
/confirmuser	Post confirmation	PostConfirmation_ConfirmSignUp
/login	Pre authentication	PreAuthentication_Authentication
	Pre token generation	TokenGeneration_Authentication TokenGeneration_AuthenticateDevice TokenGeneration_RefreshTokens
	Migrate user	UserMigration_Authentication
	Custom message	CustomMessage_Authentication
	Custom email sender	CustomEmailSender_AccountTakeOverNotification

Managed login URI	Lambda trigger	Trigger source
		`CustomEmailSender_Authentication`
	Custom SMS sender	`CustomSMSSender_Authentication`
`/forgotpassword`	Migrate user	`UserMigration_ForgotPassword`
	Custom message	`CustomMessage_ForgotPassword`
	Custom email sender	`CustomEmailSender_ForgotPassword`
	Custom SMS sender	`CustomSMSSender_ForgotPassword`
`/confirmforgotpassword`	Post confirmation	`PostConfirmation_ConfirmForgotPassword`

Lambda triggers for federated users

You can use the following Lambda triggers to customize your user pool workflows for users who sign in with a federated provider.

> **Note**
>
> Federated users can use managed login to sign in, or you can generate a request to the Authorize endpoint that silently redirects them to their identity provider sign-in page. You can't sign in federated users with the Amazon Cognito user pools API.

Federated user trigger sources

Sign-in event	Lambda trigger	Trigger source
First sign-in	Pre sign-up	`PreSignUp_ExternalProvider`
	Post confirmation	`PostConfirmation_ConfirmSignUp`
	Pre token generation	`TokenGeneration_HostedAuth`
Subsequent sign-ins	Pre authentication	`PreAuthentication_Authentication`
	Post authentication	`PostAuthentication_Authentication`
	Pre token generation	`TokenGeneration_HostedAuth`

Federated sign-in does not invoke any Custom authentication challenge Lambda triggers, Migrate user Lambda trigger, Custom message Lambda trigger, or Custom sender Lambda triggers in your user pool.

Connecting Lambda triggers to user pool functional operations

Each Lambda trigger serves a functional role in your user pool. For example, a trigger can modify your sign-up flow, or add a custom authentication challenge. The event that Amazon Cognito sends to a Lambda function can reflect one of multiple actions that make up that functional role. For example, Amazon Cognito invokes a pre sign-up trigger when your user signs up, and when you create a user. Each of these different cases for the same functional role has its own `triggerSource` value. Your Lambda function can process incoming events differently based on the operation that invoked it.

Amazon Cognito also invokes all assigned functions when an event corresponds to a trigger source. For example, when a user signs in to a user pool where you assigned migrate user and pre authentication triggers, they activate both.

Sign-up, confirmation, and sign-in (authentication) triggers

Trigger	triggerSource value	Event
Pre sign-up	`PreSignUp_SignUp`	Pre sign-up.
Pre sign-up	`PreSignUp_AdminCreateUser`	Pre sign-up when an admin creates a new user.
Pre sign-up	`PreSignUp_ExternalProvider`	Pre sign-up for external identity providers.
Post confirmation	`PostConfirmation_ConfirmSignUp`	Post sign-up confirmation.
Post confirmation	`PostConfirmation_ConfirmForgotPassword`	Post Forgot Password confirmation.
Pre authentication	`PreAuthentication_Authentication`	Pre authentication.
Post authentication	`PostAuthentication_Authentication`	Post authentication.

Custom authentication challenge triggers

Trigger	triggerSource value	Event
Define auth challenge	`DefineAuthChallenge_Authentication`	Define Auth Challenge.
Create auth challenge	`CreateAuthChallenge_Authentication`	Create Auth Challenge.
Verify auth challenge	`VerifyAuthChallengeResponse_Authentication`	Verify Auth Challenge Response.

Federation triggers

Trigger	triggerSource value	Event
Inbound federation	`InboundFederation_ExternalProvider`	Inbound federation.

Pre token generation triggers

Trigger	triggerSource value	Event
Pre token generation	`TokenGeneration_HostedAuth`	Amazon Cognito authenticates the user from your managed login sign-in page.
Pre token generation	`TokenGeneration_Authentication`	User authentication or token refresh complete.
Pre token generation	`TokenGeneration_NewPasswordChallenge`	Admin creates the user. Amazon Cognito invokes this when the user must change a temporary password.
Pre token generation	`TokenGeneration_AuthenticateDevice`	End of the authentication of a user device.
Pre token generation	`TokenGeneration_RefreshTokens`	User tries to refresh the identity and access tokens.

Migrate user triggers

Trigger	triggerSource value	Event
User migration	`UserMigration_Authentication`	User migration at the time of sign-in.
User migration	`UserMigration_ForgotPassword`	User migration during the forgot-password flow.

Custom message triggers

Trigger	triggerSource value	Event
Custom message	`CustomMessage_SignUp`	Custom message when a user signs up in your user pool.
Custom message	`CustomMessage_AdminCreateUser`	Custom message when you create a user as an administrator and Amazon Cognito sends them a temporary password.
Custom message	`CustomMessage_ResendCode`	Custom message when your existing user requests a new confirmation code.
Custom message	`CustomMessage_ForgotPassword`	Custom message when your user requests a password reset.
Custom message	`CustomMessage_UpdateUserAttribute`	Custom message when a user changes their email address or phone number and Amazon Cognito sends a verification code.
Custom message	`CustomMessage_VerifyUserAttribute`	Custom message when a user adds an email address or phone number and Amazon Cognito sends a verification code.
Custom message	`CustomMessage_Authentication`	Custom message when a user who has configured SMS MFA signs in.

Custom sender triggers

Trigger	triggerSource value	Event

Custom sender	`CustomEmailSender_SignUp`
	
	`CustomSmsSender_SignUp`
		When a user signs up in your user pool.

Custom sender	`CustomEmailSender_AdminCreateUser`
	
	`CustomSmsSender_AdminCreateUser`
		When you create a user as an administrator and Amazon Cognito sends them a temporary password.

Custom sender	`CustomEmailSender_ForgotPassword`
	
	`CustomSmsSender_ForgotPassword`
		When your user requests a password reset.

Custom sender	`CustomEmailSender_UpdateUserAttribute`
	
	`CustomSmsSender_UpdateUserAttribute`
		When a user changes their email address or phone number and Amazon Cognito sends a verification code.

Custom sender	`CustomEmailSender_VerifyUserAttribute`
	
	`CustomSmsSender_VerifyUserAttribute`
		When a user adds an email address or phone number and Amazon Cognito sends a verification code.

Custom sender	`CustomEmailSender_Authentication`
	
	`CustomSmsSender_Authentication`
		When a user who has configured SMS or email MFA or OTP signs in.

Trigger	triggerSource value	Event
Custom sender	`CustomEmailSender_AccountTakeOverNotification`	When your threat protection settings take an automated action against a user's sign-in attempt and the action for the risk level includes a notification.

Pre sign-up Lambda trigger

You might want to customize the sign-up process in user pools that have self-service sign-up options. Some common uses of the pre sign-up trigger are to perform custom analysis and recording of new users, apply security and governance standards, or link users from a third-party IdP to a consolidated user profile. You might also have trusted users who aren't required to undergo verification and confirmation.

Immediately before Amazon Cognito completes creation of a new local or federated user, it activates the pre sign-up Lambda function. The `userAttributes` in the request object sent to this function contain attributes that have been provided by local user sign-up or that have successfully been mapped from provider attributes for a federated user. Your user pool invokes this trigger on self-service sign-up with SignUp or first sign-in with a trusted identity provider, and on user creation with AdminCreateUser. As part of the sign-up process, you can use this function to analyze the sign-in event with custom logic, and modify or deny the new user.

Topics

- Pre sign-up Lambda trigger parameters
- Pre sign-up example: Auto-confirm users from a registered domain
- Pre sign-up example: Auto-confirm and auto-verify all users
- Pre sign-up example: Deny sign-up if user name has fewer than five characters

Pre sign-up Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "validationData": {
            "string": "string",
            . . .
         },
        "clientMetadata": {
            "string": "string",
            . . .
         }
    },

    "response": {
        "autoConfirmUser": "boolean",
        "autoVerifyPhone": "boolean",
        "autoVerifyEmail": "boolean"
    }
}
```

Pre sign-up request parameters

userAttributes

One or more name-value pairs representing user attributes. The attribute names are the keys.

validationData

One or more key-value pairs with user attribute data that your app passed to Amazon Cognito in the request to create a new user. Send this information to your Lambda function in the ValidationData parameter of your AdminCreateUser or SignUp API request.

Amazon Cognito doesn't set your ValidationData data as attributes of the user that you create. ValidationData is temporary user information that you supply for the purposes of your pre sign-up Lambda trigger.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the pre sign-up trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the following API actions: AdminCreateUser, AdminRespondToAuthChallenge, ForgotPassword, and SignUp.

Pre sign-up response parameters

In the response, you can set `autoConfirmUser` to `true` if you want to auto-confirm the user. You can set `autoVerifyEmail` to `true` to auto-verify the user's email. You can set `autoVerifyPhone` to `true` to auto-verify the user's phone number.

> **Note**
>
> Response parameters `autoVerifyPhone`, `autoVerifyEmail` and `autoConfirmUser` are ignored by Amazon Cognito when the pre sign-up Lambda function is triggered by the `AdminCreateUser` API.

autoConfirmUser

Set to `true` to auto-confirm the user, or `false` otherwise.

autoVerifyEmail

Set to `true` to set as verified the email address of a user who is signing up, or `false` otherwise. If `autoVerifyEmail` is set to `true`, the `email` attribute must have a valid, non-null value. Otherwise an error will occur and the user will not be able to complete sign-up.

If the `email` attribute is selected as an alias, an alias will be created for the user's email address when `autoVerifyEmail` is set. If an alias with that email address already exists, the alias will be moved to the new user and the previous user's email address will be marked as unverified. For more information, see Customizing sign-in attributes.

autoVerifyPhone

Set to `true` to set as verified the phone number of a user who is signing up, or `false` otherwise. If `autoVerifyPhone` is set to `true`, the phone_number attribute must have a valid, non-null value. Otherwise an error will occur and the user will not be able to complete sign-up.

If the phone_number attribute is selected as an alias, an alias will be created for the user's phone number when autoVerifyPhone is set. If an alias with that phone number already exists, the alias will be moved to the new user and the previous user's phone number will be marked as unverified. For more information, see Customizing sign-in attributes.

Pre sign-up example: Auto-confirm users from a registered domain

This is example Lambda trigger code. The pre sign-up trigger is invoked immediately before Amazon Cognito processes the sign-up request. It uses a custom attribute **custom:domain** to automatically confirm new users from a particular email domain. Any new users not in the custom domain will be added to the user pool, but not automatically confirmed.

Node.js

```
export const handler = async (event, context, callback) => {
  // Set the user pool autoConfirmUser flag after validating the email domain
  event.response.autoConfirmUser = false;

  // Split the email address so we can compare domains
  var address = event.request.userAttributes.email.split("@");

  // This example uses a custom attribute "custom:domain"
  if (event.request.userAttributes.hasOwnProperty("custom:domain")) {
    if (event.request.userAttributes["custom:domain"] === address[1]) {
      event.response.autoConfirmUser = true;
    }
  }

  // Return to Amazon Cognito
  callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    # It sets the user pool autoConfirmUser flag after validating the email domain
    event['response']['autoConfirmUser'] = False

    # Split the email address so we can compare domains
    address = event['request']['userAttributes']['email'].split('@')
```

```
    # This example uses a custom attribute 'custom:domain'
    if 'custom:domain' in event['request']['userAttributes']:
        if event['request']['userAttributes']['custom:domain'] == address[1]:
            event['response']['autoConfirmUser'] = True

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "request": {
        "userAttributes": {
            "email": "testuser@example.com",
            "custom:domain": "example.com"
        }
    },
    "response": {}
}
```

Pre sign-up example: Auto-confirm and auto-verify all users

This example confirms all users and sets the user's `email` and phone_number attributes to verified if the attribute is present. Also, if aliasing is enabled, aliases will be created for phone_number and `email` when auto-verify is set.

> **Note**
>
> If an alias with the same phone number already exists, the alias will be moved to the new user, and the previous user's phone_number will be marked as unverified. The same is true for email addresses. To prevent this from happening, you can use the user pools ListUsers API to see if there is an existing user who is already using the new user's phone number or email address as an alias.

Node.js

```
exports.handler = (event, context, callback) => {
  // Confirm the user
  event.response.autoConfirmUser = true;

  // Set the email as verified if it is in the request
  if (event.request.userAttributes.hasOwnProperty("email")) {
    event.response.autoVerifyEmail = true;
  }

  // Set the phone number as verified if it is in the request
  if (event.request.userAttributes.hasOwnProperty("phone_number")) {
    event.response.autoVerifyPhone = true;
  }

  // Return to Amazon Cognito
  callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    # Confirm the user
    event['response']['autoConfirmUser'] = True

    # Set the email as verified if it is in the request
    if 'email' in event['request']['userAttributes']:
        event['response']['autoVerifyEmail'] = True

    # Set the phone number as verified if it is in the request
    if 'phone_number' in event['request']['userAttributes']:
        event['response']['autoVerifyPhone'] = True

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "request": {
    "userAttributes": {
      "email": "user@example.com",
      "phone_number": "+12065550100"
    }
  },
  "response": {}
}
```

Pre sign-up example: Deny sign-up if user name has fewer than five characters

This example checks the length of the user name in a sign-up request. The example returns an error if the user has entered a name less than five characters long.

Node.js

```
export const handler = (event, context, callback) => {
    // Impose a condition that the minimum length of the username is 5 is imposed on
 all user pools.
    if (event.userName.length < 5) {
        var error = new Error("Cannot register users with username less than the
 minimum length of 5");
        // Return error to Amazon Cognito
        callback(error, event);
    }
    // Return to Amazon Cognito
    callback(null, event);
};
```

Python

```
def lambda_handler(event, context):
    if len(event['userName']) < 5:
        raise Exception("Cannot register users with username less than the minimum
 length of 5")
    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "userName": "rroe",
  "response": {}
}
```

Post confirmation Lambda trigger

Amazon Cognito invokes this trigger after a signed-up user confirms their user account. In your post confirmation Lambda function, you can send custom messages or add custom API requests. For example, you can query an external system and populate additional attributes to the user. Amazon Cognito invokes this trigger only for user who sign up in your user pool, not for user accounts that you create with your administrator credentials.

The request contains the current attributes for the confirmed user. Your user pool invokes your post confirmation function on ConfirmSignUp, AdminConfirmSignUp, and ConfirmForgotPassword. This trigger also runs when users confirm sign-up or password reset in managed login.

Topics

- Post confirmation Lambda trigger parameters
- Post confirmation example

Post confirmation Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
            "userAttributes": {
```

```
                "string": "string",
                . . .
            },
            "clientMetadata": {
             "string": "string",
             . . .
            }
        },
    "response": {}
}
```

Post confirmation request parameters

userAttributes

One or more key-value pairs representing user attributes.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the post confirmation trigger. You can pass this data to your Lambda function by using the ClientMetadata parameter in the following API actions: AdminConfirmSignUp, ConfirmForgotPassword, ConfirmSignUp, and SignUp.

Post confirmation response parameters

No additional return information is expected in the response.

Post confirmation example

This example Lambda function sends a confirmation email message to your user using Amazon SES. For more information see Amazon Simple Email Service Developer Guide.

Node.js

```
// Import required AWS SDK clients and commands for Node.js. Note that this requires
// the `@aws-sdk/client-ses` module to be either bundled with this code or included
// as a Lambda layer.
import { SES, SendEmailCommand } from "@aws-sdk/client-ses";
const ses = new SES();

const handler = async (event) => {
```

```
  if (event.request.userAttributes.email) {
    await sendTheEmail(
      event.request.userAttributes.email,
      `Congratulations ${event.userName}, you have been confirmed.`,
    );
  }
  return event;
};

const sendTheEmail = async (to, body) => {
  const eParams = {
    Destination: {
      ToAddresses: [to],
    },
    Message: {
      Body: {
        Text: {
          Data: body,
        },
      },
      Subject: {
        Data: "Cognito Identity Provider registration completed",
      },
    },
    // Replace source_email with your SES validated email address
    Source: "<source_email>",
  };
  try {
    await ses.send(new SendEmailCommand(eParams));
  } catch (err) {
    console.log(err);
  }
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "request": {
        "userAttributes": {
            "email": "user@example.com",
            "email_verified": true
        }
    },
    "response": {}
}
```

Pre authentication Lambda trigger

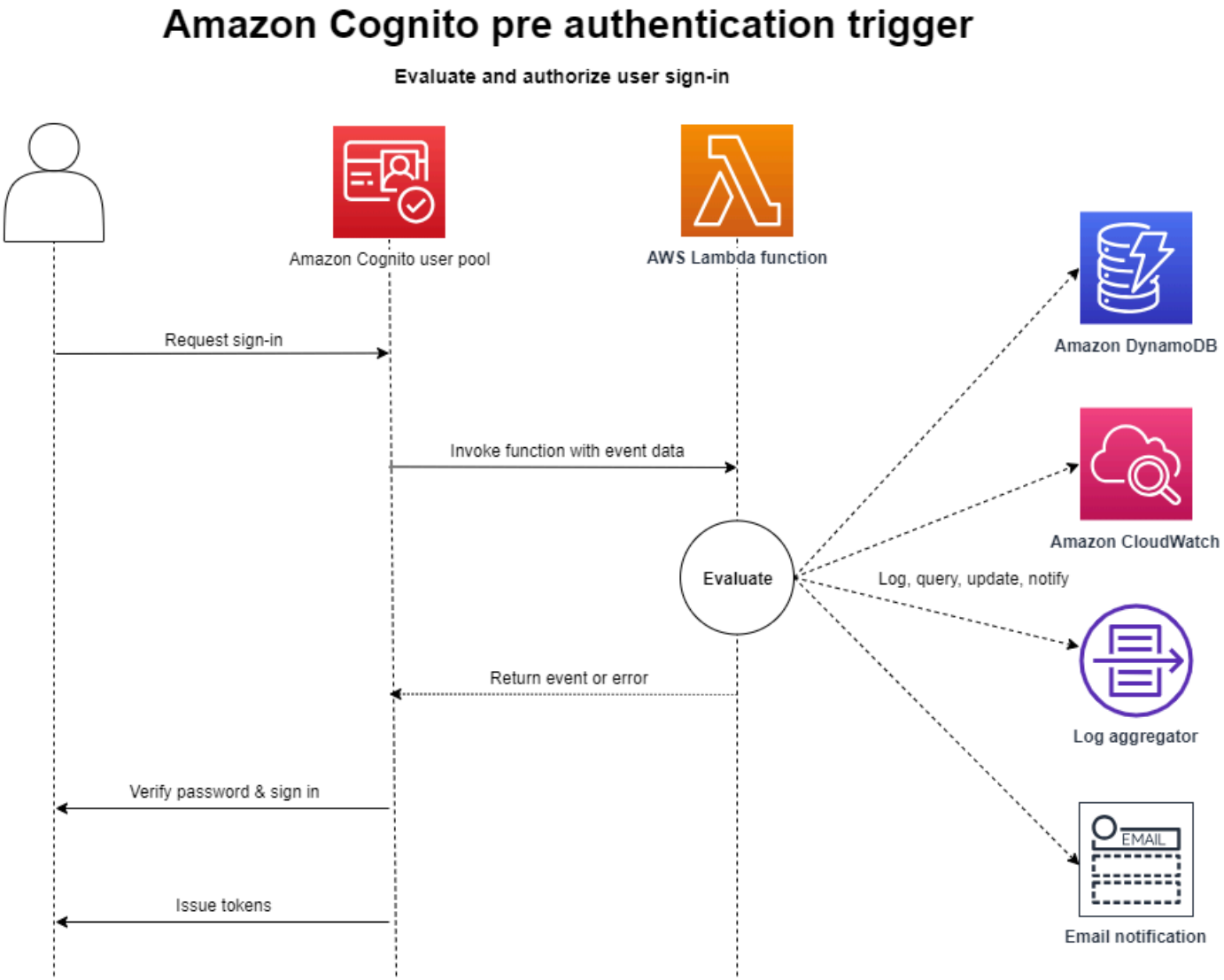
Amazon Cognito invokes this trigger when a user attempts to sign in so that you can create custom validation that performs preparatory actions. For example, you can deny the authentication request or record session data to an external system.

> **Note**
>
> This Lambda trigger doesn't activate when a user doesn't exist unless the `PreventUserExistenceErrors` setting of a user pool app client is set to `ENABLED`. Renewal of an existing authentication session also doesn't activate this trigger.

Topics

- Flow overview
- Pre authentication Lambda trigger parameters
- Pre authentication example

Flow overview



The request includes client validation data from the `ClientMetadata` values that your app passes to the user pool `InitiateAuth` and `AdminInitiateAuth` API operations.

For more information, see An example authentication session.

Pre authentication Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "validationData": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {}
}
```

Pre authentication request parameters

userAttributes

One or more name-value pairs that represent user attributes.

userNotFound

When you set `PreventUserExistenceErrors` to `ENABLED` for your user pool client, Amazon Cognito populates this Boolean.

validationData

One or more key-value pairs that contain the validation data in the user's sign-in request. To pass this data to your Lambda function, use the ClientMetadata parameter in the InitiateAuth and AdminInitiateAuth API actions.

Pre authentication response parameters

Amazon Cognito doesn't process any added information that your function returns in the response. Your function can return an error to reject the sign-in attempt, or use API operations to query and modify your resources.

Pre authentication example

This example function prevents users from signing in to your user pool with a specific app client. Because the pre authentication Lambda function doesn't invoke when your user has an existing session, this function only prevents *new* sessions with the app client ID that you want to block.

Node.js

```
const handler = async (event) => {
  if (
    event.callerContext.clientId === "user-pool-app-client-id-to-be-blocked"
  ) {
    throw new Error("Cannot authenticate users from this user pool app client");
  }

  return event;
};

export { handler };
```

Python

```
def lambda_handler(event, context):
    if event['callerContext']['clientId'] == "<user pool app client id to be
 blocked>":
        raise Exception("Cannot authenticate users from this user pool app client")

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "callerContext": {
        "clientId": "<user pool app client id to be blocked>"
```

```
    },
    "response": {}
}
```

Post authentication Lambda trigger

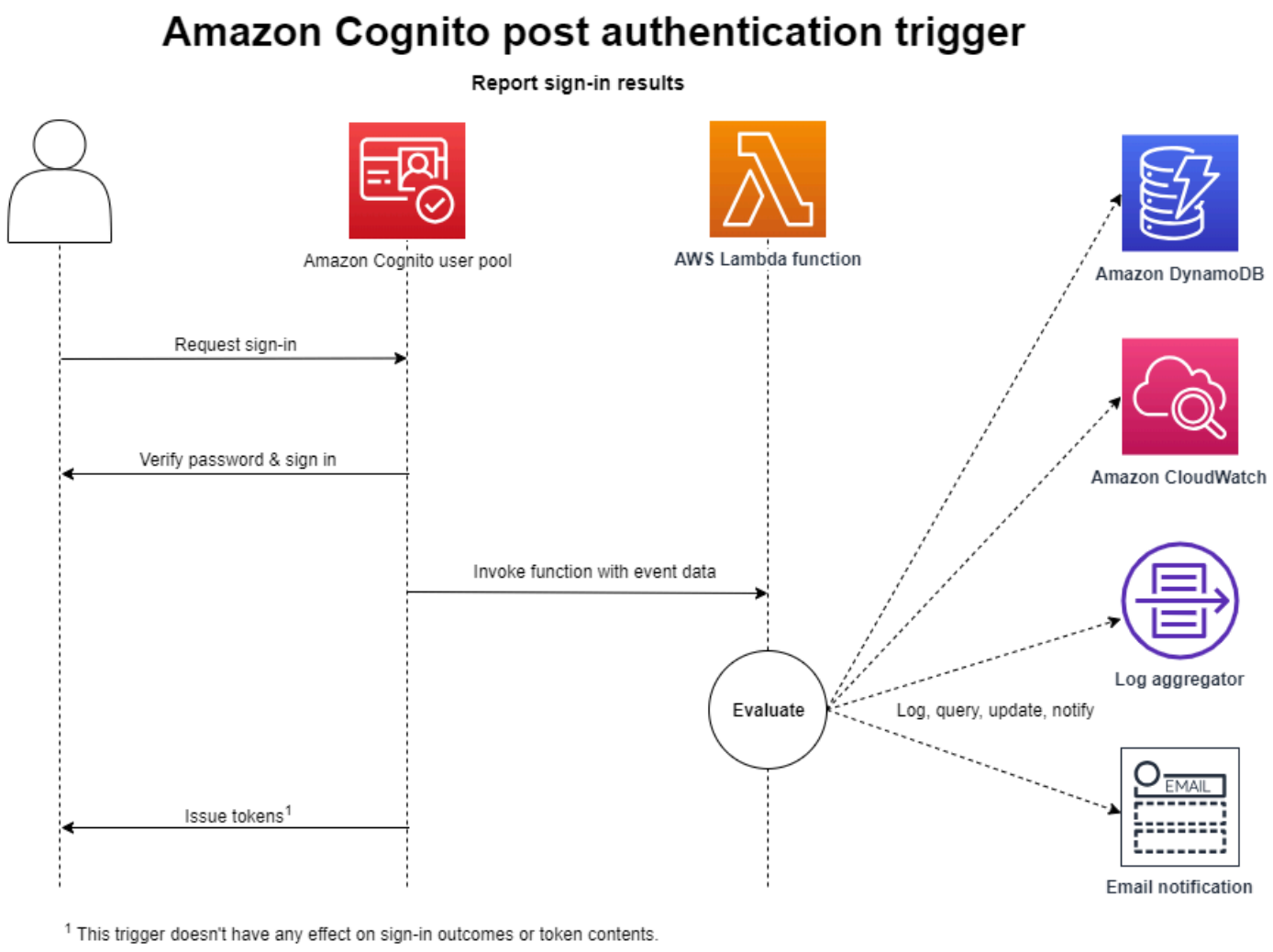
The post authentication trigger doesn't change the authentication flow for a user. Amazon Cognito invokes this Lambda after authentication is complete, before a user has received tokens. Add a post authentication trigger when you want to add custom post-processing of authentication events, for example logging or user profile adjustments that will be reflected on the next sign-in.

A post authentication Lambda that doesn't return the request body to Amazon Cognito can still cause authentication to fail to complete. For more information, see Things to know about Lambda triggers.

Topics

- Authentication flow overview
- Post authentication Lambda trigger parameters
- Post authentication example

Authentication flow overview



For more information, see An example authentication session.

Post authentication Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
```

```
            "string": "string",
            . . .
        },
        "newDeviceUsed": boolean,
        "clientMetadata": {
            "string": "string",
            . . .
           }
       },
    "response": {}
}
```

Post authentication request parameters

newDeviceUsed

This flag indicates if the user has signed in on a new device. Amazon Cognito only sets this flag if the remembered devices value of the user pool is `Always` or `User Opt-In`.

userAttributes

One or more name-value pairs representing user attributes.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the post authentication trigger. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions. Amazon Cognito doesn't include data from the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the post authentication function.

Post authentication response parameters

Amazon Cognito doesn't expect any additional return information in the response. Your function can use API operations to query and modify your resources, or record event metadata to an external system.

Post authentication example

This post authentication sample Lambda function sends data from a successful sign-in to CloudWatch Logs.

Node.js

```
const handler = async (event) => {
  // Send post authentication data to Amazon CloudWatch logs
  console.log("Authentication successful");
  console.log("Trigger function =", event.triggerSource);
  console.log("User pool = ", event.userPoolId);
  console.log("App client ID = ", event.callerContext.clientId);
  console.log("User ID = ", event.userName);

  return event;
};

export { handler };
```

Python

```
import os
def lambda_handler(event, context):

    # Send post authentication data to Cloudwatch logs
    print ("Authentication successful")
    print ("Trigger function =", event['triggerSource'])
    print ("User pool = ", event['userPoolId'])
    print ("App client ID = ", event['callerContext']['clientId'])
    print ("User ID = ", event['userName'])

    # Return to Amazon Cognito
    return event
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "triggerSource": "testTrigger",
  "userPoolId": "testPool",
  "userName": "testName",
```

```
  "callerContext": {
      "clientId": "12345"
  },
  "response": {}
}
```

Inbound federation Lambda trigger

The inbound federation trigger transforms federated user attributes during the authentication process with external identity providers. When users authenticate through configured identity providers, this trigger allows you to modify responses from external SAML and OIDC providers by intercepting and transforming data in the authentication process, providing programmatic control over how Amazon Cognito user pools handle federated users and their attributes.

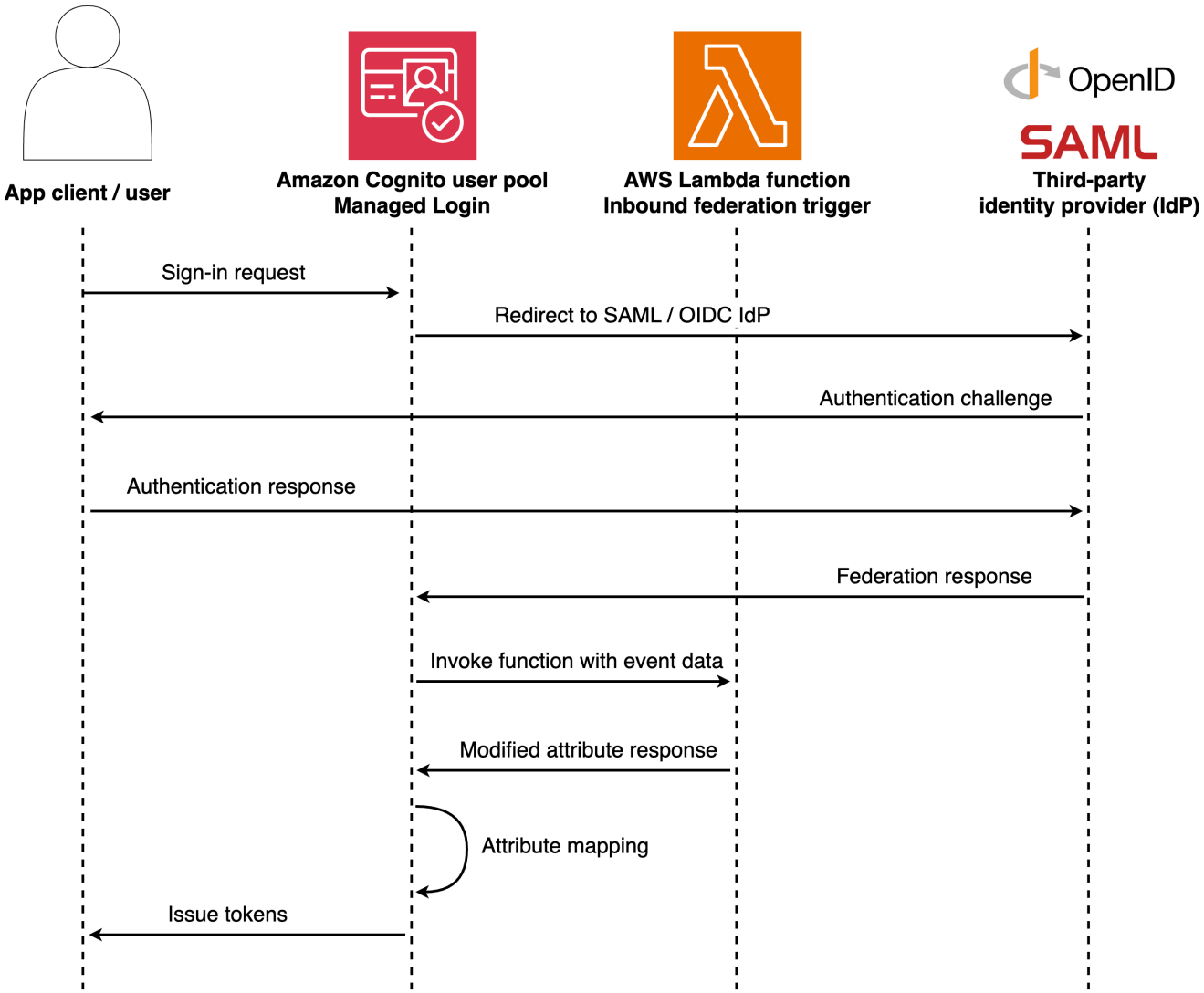
Use this trigger to add, override, or suppress attributes before creating new users or updating existing federated user profiles. This trigger receives raw identity provider attributes as input and returns modified attributes that Amazon Cognito applies to the user profile.

Topics

- Flow overview
- Inbound federation Lambda trigger parameters
- Inbound federation example: Group membership management
- Inbound federation example: Truncate large attributes
- Inbound federation example: Logging federation events

Flow overview

When a user authenticates with an external identity provider, Amazon Cognito invokes the inbound federation trigger before creating or updating the user profile. The trigger receives the raw attributes from the identity provider and can transform them before Amazon Cognito stores them. This flow occurs for both new federated users and existing users who sign in again through federation.



Inbound federation Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "version": "string",
    "triggerSource": "InboundFederation_ExternalProvider",
    "region": AWSRegion,
    "userPoolId": "string",
    "userName": "string",
```

```
    "callerContext": {
        "awsSdkVersion": "string",
        "clientId": "string"
    },
    "request": {
        "providerName": "string",
        "providerType": "string",
        "attributes": {
            "tokenResponse": {
                "access_token": "string",
                "token_type": "string",
                "expires_in": "string"
            },
            "idToken": {
                "sub": "string",
                "email": "string",
                "email_verified": "string"
            },
            "userInfo": {
                "email": "string",
                "given_name": "string",
                "family_name": "string"
            },
            "samlResponse": {
                "string": "string"
            }
        }
    },
    "response": {
        "userAttributesToMap": {
            "string": "string"
        }
    }
}
```

Inbound federation request parameters

providerName

The name of the external identity provider.

providerType

The type of the external identity provider. Valid values: `OIDC`, `SAML`, `Facebook`, `Google`, `SignInWithApple`, `LoginWithAmazon`.

attributes

The raw attributes received from the identity provider before processing. The structure varies based on provider type.

attributes.tokenResponse

OAuth token response data from the `/token` endpoint. Available for OIDC and social providers only. Contains `access_token`, `id_token`, `refresh_token`, `token_type`, `expires_in`, and `scope`.

attributes.idToken

Decoded and validated ID token JWT claims. Available for OIDC and social providers only. Contains verified user identity information including `sub` (unique user identifier), `email`, `name`, `iss` (issuer), `aud` (audience), `exp` (expiration), and `iat` (issued time).

attributes.userInfo

Extended user profile information from the UserInfo endpoint. Available for OIDC and social providers only. Contains detailed profile attributes such as `given_name`, `family_name`, `picture`, `address`, and other provider-specific fields. May be empty if the IdP doesn't support the UserInfo endpoint or if the endpoint call fails.

attributes.samlResponse

SAML assertion attributes. Available for SAML providers only. Contains attributes from the SAML response.

Inbound federation response parameters

userAttributesToMap

The user attributes to apply to the user profile.

Important

You must include ALL user attributes you want to retain in the response, including attributes you are not modifying. Any attributes not included in the `userAttributesToMap` response will be dropped and not stored in the user profile. This applies to both modified and unmodified attributes.

Empty response behavior

If you return an empty object `{}` for `userAttributesToMap`, all original attributes from the identity provider are retained unchanged. This acts as a no-op, as if the Lambda function was never executed. This is different from omitting attributes, which drops them.

Provider-specific attributes

The structure of `request.attributes` varies based on the `providerType`. OIDC and social providers include `tokenResponse`, `idToken`, and `userInfo` objects. SAML providers include only the `samlResponse` object.

Inbound federation example: Group membership management

This example shows how to map federated identity provider groups to Amazon Cognito user pools groups. This function extracts group membership from the federated response and automatically adds users to corresponding Amazon Cognito groups, eliminating the need for post-authentication triggers.

Node.js

```
exports.handler = async (event) => {
    const { providerType, attributes } = event.request;

    // Extract user attributes based on provider type
    let userAttributesFromIdp = {};
    if (providerType === 'SAML') {
        userAttributesFromIdp = attributes.samlResponse || {};
    } else {
```

```
        // For OIDC and Social providers, merge userInfo and idToken
        userAttributesFromIdp = {
            ...(attributes.userInfo || {}),
            ...(attributes.idToken || {})
        };
    }

    // Extract groups from federated response
    const federatedGroups = userAttributesFromIdp.groups?.split(',') || [];

    // Map federated groups to Cognito groups
    const groupMapping = {
        'Domain Admins': 'Administrators',
        'Engineering': 'Developers',
        'Sales': 'SalesTeam'
    };

    // Filter to only in-scope groups
    const mappedGroups = federatedGroups
        .map(group => groupMapping[group.trim()])
        .filter(group => group); // Remove undefined values

    // Pass through attributes with mapped groups as custom attribute
    const attributesToMap = {
        ...userAttributesFromIdp,
        'custom:user_groups': mappedGroups.join(',')
    };

    // Remove original groups attribute
    delete attributesToMap.groups;

    event.response.userAttributesToMap = attributesToMap;
    return event;
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "userPoolId": "us-east-1_XXXXXXXXX",
    "request": {
        "providerName": "CorporateAD",
        "providerType": "SAML",
        "attributes": {
            "samlResponse": {
                "email": "jane.smith@company.com",
                "given_name": "Jane",
                "family_name": "Smith",
                "groups": "Engineering,Domain Admins",
                "department": "Engineering"
            }
        }
    },
    "response": {
        "userAttributesToMap": {}
    }
}
```

Inbound federation example: Truncate large attributes

This example shows how to truncate attribute values that exceed Amazon Cognito's storage limits. This function checks each attribute from the identity provider. If an attribute value exceeds 2048 characters, it truncates the value and adds ellipsis to indicate truncation. All other attributes pass through unchanged.

Node.js

```
exports.handler = async (event) => {
    const MAX_ATTRIBUTE_LENGTH = 2048;

    // Get the identity provider attributes based on provider type
    const { providerType, attributes } = event.request;
    let idpAttributes = {};

    if (providerType === 'SAML') {
        idpAttributes = attributes.samlResponse || {};
    } else {
        // For OIDC and Social providers, merge userInfo and idToken
```

```
        idpAttributes = {
            ...(attributes.userInfo || {}),
            ...(attributes.idToken || {})
        };
    }

    const userAttributes = {};

    // Process each attribute
    for (const [key, value] of Object.entries(idpAttributes)) {
        if (typeof value === 'string' && value.length > MAX_ATTRIBUTE_LENGTH) {
            // Truncate the value and add ellipsis
            userAttributes[key] = value.substring(0, MAX_ATTRIBUTE_LENGTH - 3) +
 '...';
            console.log(`Truncated attribute ${key} from ${value.length} to
 ${userAttributes[key].length} characters`);
        } else {
            // Keep the original value
            userAttributes[key] = value;
        }
    }

    // Return the modified attributes
    event.response.userAttributesToMap = userAttributes;
    return event;
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "version": "string",
    "triggerSource": "InboundFederation_ExternalProvider",
    "region": "us-east-1",
    "userPoolId": "us-east-1_XXXXXXXXX",
    "userName": "ExampleProvider_12345",
    "callerContext": {
        "awsSdkVersion": "string",
```

```
        "clientId": "string"
    },
    "request": {
        "providerName": "ExampleProvider",
        "providerType": "OIDC",
        "attributes": {
            "tokenResponse": {
                "access_token": "abcDE...",
                "token_type": "Bearer",
                "expires_in": "3600"
            },
            "idToken": {
                "sub": "12345",
                "email": "user@example.com"
            },
            "userInfo": {
                "email": "user@example.com",
                "given_name": "Example",
                "family_name": "User",
                "bio": "This is a very long biography that contains more than 2048
 characters..."
            }
        }
    },
    "response": {
        "userAttributesToMap": {}
    }
}
```

Inbound federation example: Logging federation events

This example shows how to log federated authentication events for monitoring and debugging. This example function captures detailed information about federated users and their attributes, providing visibility into the authentication process.

Node.js

```
exports.handler = async (event) => {
    const { providerName, providerType, attributes } = event.request;

    // Extract user attributes based on provider type
    let userAttributesFromIdp = {};
```

```
    if (providerType === 'SAML') {
        userAttributesFromIdp = attributes.samlResponse || {};
    } else {
        // For OIDC and Social providers, merge userInfo and idToken
        userAttributesFromIdp = {
            ...(attributes.userInfo || {}),
            ...(attributes.idToken || {})
        };
    }

    // Log federated authentication details
    console.log(JSON.stringify({
        timestamp: new Date().toISOString(),
        providerName,
        providerType,
        userEmail: userAttributesFromIdp.email,
        attributeCount: Object.keys(userAttributesFromIdp).length,
        attributes: userAttributesFromIdp
    }));

    // Pass through all attributes unchanged
    event.response.userAttributesToMap = userAttributesFromIdp;
    return event;
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "version": "string",
    "triggerSource": "InboundFederation_ExternalProvider",
    "region": "us-east-1",
    "userPoolId": "us-east-1_XXXXXXXXX",
    "userName": "CorporateAD_john.doe",
    "callerContext": {
        "awsSdkVersion": "string",
        "clientId": "string"
    },
```

```
    "request": {
        "providerName": "CorporateAD",
        "providerType": "SAML",
        "attributes": {
            "samlResponse": {
                "email": "john.doe@company.com",
                "given_name": "John",
                "family_name": "Doe",
                "department": "Engineering",
                "employee_id": "EMP12345"
            }
        }
    },
    "response": {
        "userAttributesToMap": {}
    }
}
```

Expected CloudWatch Logs output:

JSON

```
{
    "timestamp": "2025-01-14T21:17:40.153Z",
    "providerName": "CorporateAD",
    "providerType": "SAML",
    "userEmail": "john.doe@company.com",
    "attributeCount": 5,
    "attributes": {
        "email": "john.doe@company.com",
        "given_name": "John",
        "family_name": "Doe",
        "department": "Engineering",
        "employee_id": "EMP12345"
    }
}
```

Custom authentication challenge Lambda triggers

As you build out your authentication flows for your Amazon Cognito user pool, you might find that you want to extend your authentication model beyond the built-in flows. One common

use case for the custom challenge triggers is to implement additional security checks beyond username, password, and multi-factor authentication (MFA). A custom challenge is any question and response you can generate in a Lambda-supported programming language. For example, you might want to require users to solve a CAPTCHA or answer a security question before being allowed to authenticate. Another potential need is to integrate with specialized authentication factors or devices. Or you might have already developed software that authenticates users with a hardware security key or a biometric device. The definition of authentication success for a custom challenge is whatever answer your Lambda function accepts as correct: a fixed string, for example, or a satisfactory response from an external API.

You can start authentication with your custom challenge and control the authentication process entirely, or you can perform username-password authentication before your application receives your custom challenge.

The custom authentication challenge Lambda trigger:

Defines

Initiates a challenge sequence. Determines whether you want to initiate a new challenge, mark authentication as complete, or halt the authentication attempt.

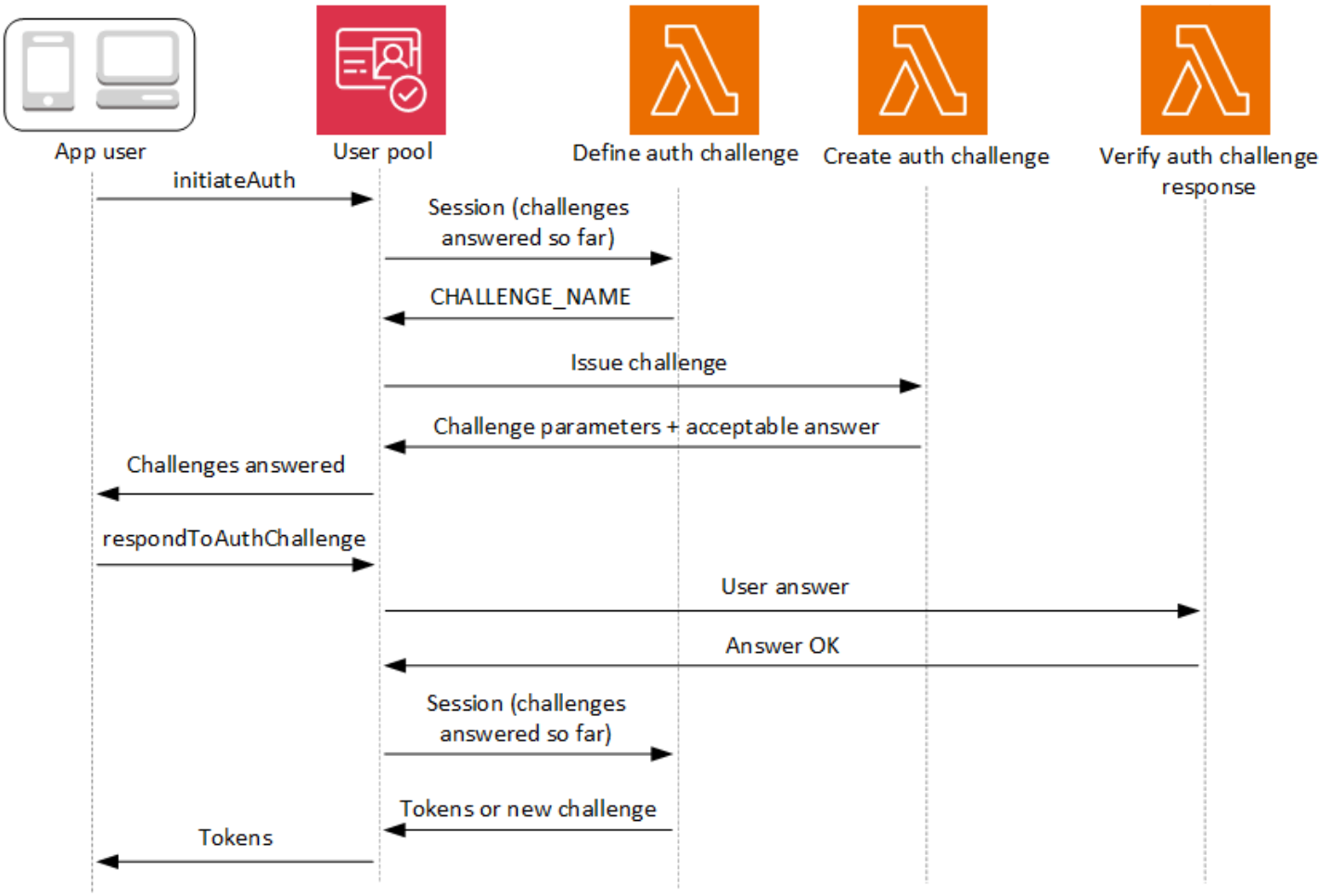
Creates

Issues the question to your application that the user must answer. This function might present a security question or a link to a CAPTCHA that your application should display to your user.

Verifies

Knows the expected answer and compares it to the answer your application provides in the challenge response. The function might call the API of your CAPTCHA service to retrieve the expected results of your user's attempted solution.

These three Lambda functions chain together to present an authentication mechanism that is completely within your control and of your own design. Because custom authentication requires application logic in your client and in the Lambda functions, you can't process custom authentication within managed login. This authentication system requires additional developer effort. Your application must perform the authentication flow with the user pools API and handle the resulting challenge with a custom-built login interface that renders the question at the center of the custom authentication challenge.



For more information about implementing custom authentication, see Custom authentication flow and challenges

Authentication between the API operations InitiateAuth or AdminInitiateAuth, and RespondToAuthChallenge or AdminRespondToAuthChallenge. In this flow, a user authenticates by answering successive challenges until authentication either fails or the user is issued tokens. A challenge response might be a new challenge. In this case, your application responds as many times as necessary to new challenges. Successful authentication happens when the define auth challenge function analyzes the results so far, determines all challenges have been answered, and returns `IssueTokens`.

Why your custom authentication trigger didn't fire

If your Create auth challenge or Verify auth challenge response function isn't invoked, the cause is usually one of the following:

- The app client must permit a custom-authentication flow. Activate `ALLOW_CUSTOM_AUTH` (or, for choice-based sign-in, `ALLOW_USER_AUTH`) on the app client, and begin sign-in with the matching flow. A pool that only permits SRP or password authentication never enters the custom challenge sequence.
- Amazon Cognito invokes the three functions in order. It always calls Define auth challenge first, and it calls Create auth challenge and Verify auth challenge response only when your Define auth challenge function returns `challengeName: CUSTOM_CHALLENGE`. If Create isn't running, confirm that Define returns `challengeName: CUSTOM_CHALLENGE` with `issueTokens: false` and `failAuthentication: false`, rather than setting `issueTokens: true` or `failAuthentication: true`.
- All three triggers must be assigned to the user pool. A missing assignment for any one of them breaks the chain.

Topics

- SRP authentication in custom challenge flows
- Define Auth challenge Lambda trigger
- Create Auth challenge Lambda trigger
- Verify Auth challenge response Lambda trigger

SRP authentication in custom challenge flows

You can have Amazon Cognito verify user passwords before it issues your custom challenges. Any Lambda triggers associated in the Authentication category of request-rate quotas will run when you perform SRP authentication in a custom challenge flow. Here is an overview of the process:

1. Your app initiates sign-in by calling `InitiateAuth` or `AdminInitiateAuth` with the `AuthParameters` map. Parameters must include `CHALLENGE_NAME: SRP_A,` and values for `SRP_A` and `USERNAME`.
2. Amazon Cognito invokes your define auth challenge Lambda trigger with an initial session that contains `challengeName: SRP_A` and `challengeResult: true`.
3. After receiving those inputs, your Lambda function responds with `challengeName: PASSWORD_VERIFIER`, `issueTokens: false`, `failAuthentication: false`.

4. If the password verification succeeds, Amazon Cognito invokes your Lambda function again with a new session containing `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`.
5. To initiate your custom challenges, your Lambda function responds with `challengeName: CUSTOM_CHALLENGE`, `issueTokens: false`, and `failAuthentication: false`. If you don't want to start your custom auth flow with password verification, you can initiate sign-in with the `AuthParameters` map including `CHALLENGE_NAME: CUSTOM_CHALLENGE`.
6. The challenge loop repeats until all challenges are answered.

The following is an example of a starting `InitiateAuth` request that precedes custom authentication with an SRP flow.

```
{
    "AuthFlow": "CUSTOM_AUTH",
    "ClientId": "1example23456789",
    "AuthParameters": {
        "CHALLENGE_NAME": "SRP_A",
        "USERNAME": "testuser",
        "SRP_A": "[SRP_A]",
        "SECRET_HASH": "[secret hash]"
    }
}
```

Password reset in the custom authentication SRP flow

When users are in `FORCE_CHANGE_PASSWORD` status, your custom authentication flow must integrate the password change step while maintaining the integrity of your authentication challenges. Amazon Cognito invokes your define auth challenge Lambda trigger during the `NEW_PASSWORD_REQUIRED` challenge. In this scenario, a user signing in with a custom challenge flow and SRP authentication can set a new password if they are in a password-reset state.

When users are in the `RESET_REQUIRED` or `FORCE_CHANGE_PASSWORD` status, they must respond to a `NEW_PASSWORD_REQUIRED` challenge with a `NEW_PASSWORD`. In custom authentication with SRP, Amazon Cognito returns a `NEW_PASSWORD_REQUIRED` challenge after users complete the SRP `PASSWORD_VERIFIER` challenge. Your define auth challenge trigger receives both challenge results in the `session` array, and can proceed with additional custom challenges after the user successfully changes their password.

Your define auth challenge Lambda trigger must manage the challenge sequence through SRP authentication, password reset, and subsequent custom challenges. The trigger receives an array of completed challenges in the `session` parameter, including both `PASSWORD_VERIFIER` and `NEW_PASSWORD_REQUIRED` results. For an example implementation, see Define Auth challenge example.

Authentication flow steps

For users who need to verify their password before custom challenges, the process follows these steps:

1. Your app initiates sign-in by calling `InitiateAuth` or `AdminInitiateAuth` with the `AuthParameters` map. Parameters must include `CHALLENGE_NAME: SRP_A`, and values for `SRP_A` and `USERNAME`.
2. Amazon Cognito invokes your define auth challenge Lambda trigger with an initial session that contains `challengeName: SRP_A` and `challengeResult: true`.
3. After receiving those inputs, your Lambda function responds with `challengeName: PASSWORD_VERIFIER`, `issueTokens: false`, `failAuthentication: false`.
4. If the password verification succeeds, one of two things happens:

   For users in normal status:

   Amazon Cognito invokes your Lambda function again with a new session containing `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`.

   To initiate your custom challenges, your Lambda function responds with `challengeName: CUSTOM_CHALLENGE`, `issueTokens: false`, and `failAuthentication: false`.

   For users in `RESET_REQUIRED` or `FORCE_CHANGE_PASSWORD` status:

   Amazon Cognito invokes your Lambda function with a session containing `challengeName: PASSWORD_VERIFIER` and `challengeResult: true`.

   Your Lambda function should respond with `challengeName: NEW_PASSWORD_REQUIRED`, `issueTokens: false`, and `failAuthentication: false`.

   After successful password change, Amazon Cognito invokes your Lambda function with a session containing both the `PASSWORD_VERIFIER` and `NEW_PASSWORD_REQUIRED` results.

   To initiate your custom challenges, your Lambda function responds with `challengeName: CUSTOM_CHALLENGE`, `issueTokens: false`, and `failAuthentication: false`.

5. The challenge loop repeats until all challenges are answered.

If you don't want to start your custom auth flow with password verification, you can initiate sign-in with the `AuthParameters` map including `CHALLENGE_NAME: CUSTOM_CHALLENGE`.

Session management

The authentication flow maintains session continuity through a series of session IDs and challenge results. Each challenge response generates a new session ID to prevent session reuse errors, which is particularly important for multi-factor authentication flows.

Challenge results are stored chronologically in the session array that your Lambda triggers receive. For users in `FORCE_CHANGE_PASSWORD` status, the session array contains:

1. `session[0]` - Initial `SRP_A` challenge
2. `session[1]` - `PASSWORD_VERIFIER` result
3. `session[2]` - `NEW_PASSWORD_REQUIRED` result
4. Subsequent elements - Results of additional custom challenges

Example authentication flow

The following example demonstrates a complete custom authentication flow for a user in `FORCE_CHANGE_PASSWORD` status who must complete both password change and a custom CAPTCHA challenge.

1. **InitiateAuth request**

```
{
    "AuthFlow": "CUSTOM_AUTH",
    "ClientId": "1example23456789",
    "AuthParameters": {
        "CHALLENGE_NAME": "SRP_A",
        "USERNAME": "testuser",
        "SRP_A": "[SRP_A]"
    }
}
```

2. **InitiateAuth response**

```
{
```

```
    "ChallengeName": "PASSWORD_VERIFIER",
    "ChallengeParameters": {
        "USER_ID_FOR_SRP": "testuser"
    },
    "Session": "[session_id_1]"
}
```

3. **RespondToAuthChallenge request with `PASSWORD_VERIFIER`**

```
{
    "ChallengeName": "PASSWORD_VERIFIER",
    "ClientId": "1example23456789",
    "ChallengeResponses": {
        "PASSWORD_CLAIM_SIGNATURE": "[claim_signature]",
        "PASSWORD_CLAIM_SECRET_BLOCK": "[secret_block]",
        "TIMESTAMP": "[timestamp]",
        "USERNAME": "testuser"
    },
    "Session": "[session_id_1]"
}
```

4. **RespondToAuthChallenge response with `NEW_PASSWORD_REQUIRED` challenge**

```
{
    "ChallengeName": "NEW_PASSWORD_REQUIRED",
    "ChallengeParameters": {},
    "Session": "[session_id_2]"
}
```

5. **RespondToAuthChallenge request with `NEW_PASSWORD_REQUIRED`**

```
{
    "ChallengeName": "NEW_PASSWORD_REQUIRED",
    "ClientId": "1example23456789",
    "ChallengeResponses": {
        "NEW_PASSWORD": "[password]",
        "USERNAME": "testuser"
    },
    "Session": "[session_id_2]"
}
```

6. **RespondToAuthChallenge response with CAPTCHA custom challenge**

```
{
    "ChallengeName": "CUSTOM_CHALLENGE",
    "ChallengeParameters": {
        "captchaUrl": "url/123.jpg"
    },
    "Session": "[session_id_3]"
}
```

7. **RespondToAuthChallenge request with answer to CAPTCHA custom challenge**

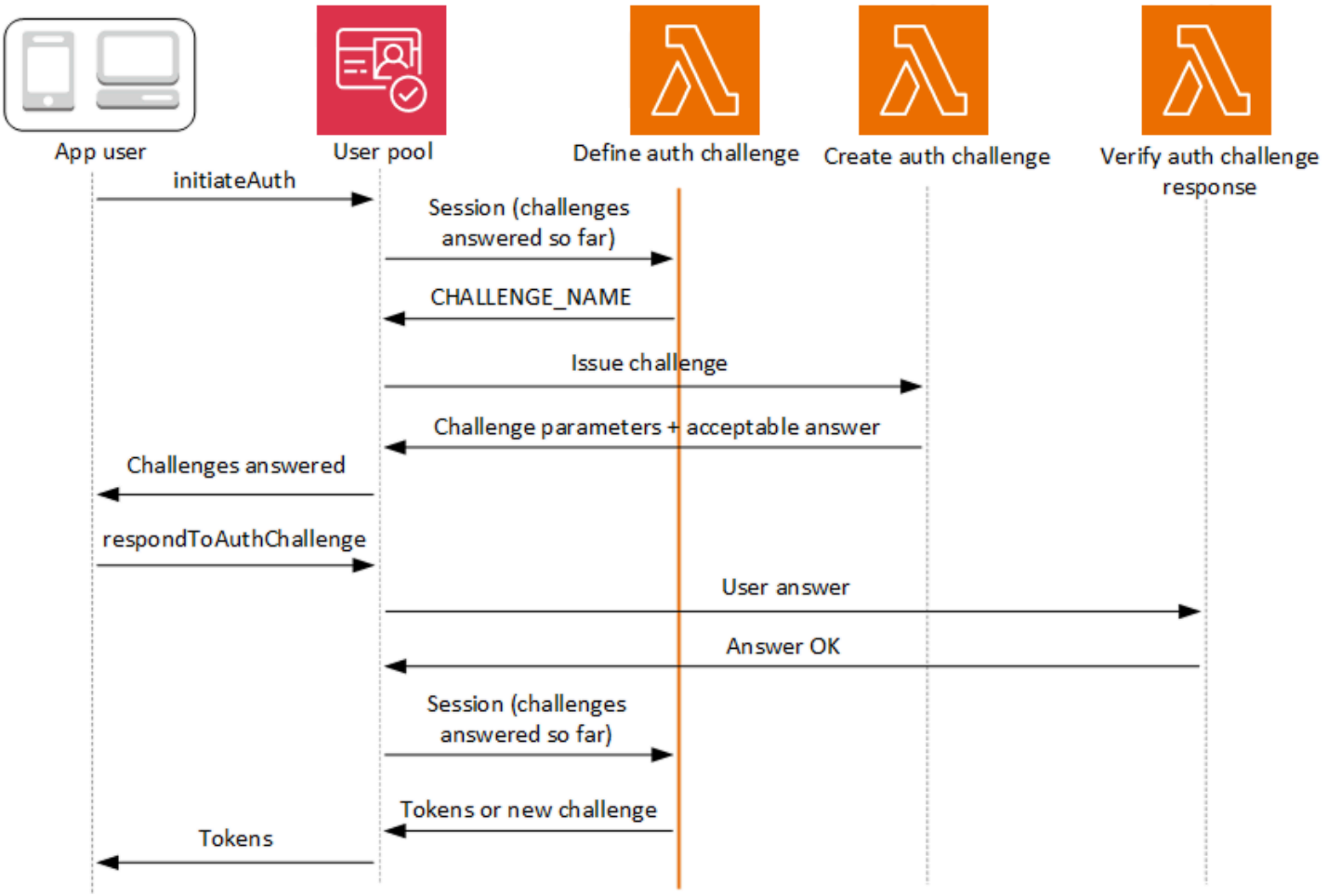
```
{
    "ChallengeName": "CUSTOM_CHALLENGE",
    "ClientId": "1example23456789",
    "ChallengeResponses": {
        "ANSWER": "123",
        "USERNAME": "testuser"
    },
    "Session": "[session_id_3]"
}
```

6. Final success response

```
{
    "AuthenticationResult": {
        "AccessToken": "eyJra456defEXAMPLE",
        "ExpiresIn": 3600,
        "IdToken": "eyJra789ghiEXAMPLE",
        "RefreshToken": "eyJjd123abcEXAMPLE",
        "TokenType": "Bearer"
    },
    "ChallengeParameters": {}
}
```

Define Auth challenge Lambda trigger

The define auth challenge trigger is a Lambda function that maintains the challenge sequence in a custom authentication flow. It declares success or failure of the challenge sequence, and sets the next challenge if the sequence isn't yet complete.



Define auth challenge

Amazon Cognito invokes this trigger to initiate the [custom authentication flow].

The request for this Lambda trigger contains `session`. The `session` parameter is an array that contains all of the challenges that are presented to the user in the current authentication process. The request also includes the corresponding result. The `session` array stores challenge details (`ChallengeResult`) in chronological order. The challenge `session[0]` represents the first challenge that the user receives.

Topics

- [Define Auth challenge Lambda trigger parameters]
- [Define Auth challenge example]

Define Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
                . . .
        },
        "session": [
            ChallengeResult,
            . . .
        ],
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "challengeName": "string",
        "issueTokens": boolean,
        "failAuthentication": boolean
    }
}
```

Define Auth challenge request parameters

When Amazon Cognito invokes your Lambda function, Amazon Cognito provides the following parameters:

userAttributes

One or more name-value pairs that represent user attributes.

userNotFound

A Boolean that Amazon Cognito populates when `PreventUserExistenceErrors` is set to ENABLED for your user pool client. A value of `true` means that the user id

(username, email address, and other details) did not match any existing users. When `PreventUserExistenceErrors` is set to `ENABLED`, the service doesn't inform the app of nonexistent users. We recommend that your Lambda functions maintain the same user experience and account for latency. This way, the caller can't detect different behavior when the user exists or doesn't exist.

session

An array of `ChallengeResult` elements. Each contains the following elements:

challengeName

One of the following challenge types: `CUSTOM_CHALLENGE`, `SRP_A`, `PASSWORD_VERIFIER`, `SMS_MFA`, `EMAIL_OTP`, `SOFTWARE_TOKEN_MFA`, `DEVICE_SRP_AUTH`, `DEVICE_PASSWORD_VERIFIER`, or `ADMIN_NO_SRP_AUTH`.

When your define auth challenge function issues a `PASSWORD_VERIFIER` challenge for a user who has set up multifactor authentication, Amazon Cognito follows it up with an `SMS_MFA`, `EMAIL_OTP`, or `SOFTWARE_TOKEN_MFA` challenge. These are the prompts for a multi-factor authentication code. In your function, include handling for input events from `SMS_MFA`, `EMAIL_OTP`, and `SOFTWARE_TOKEN_MFA` challenges. You don't need to invoke any MFA challenges in your define auth challenge function.

> **Important**
>
> When your function is determining whether a user has successfully authenticated and you should issue them tokens, always check `challengeName` in your define auth challenge function and verify that it matches the expected value.

challengeResult

Set to `true` if the user successfully completed the challenge, or `false` otherwise.

challengeMetadata

Your name for the custom challenge. Used only if `challengeName` is `CUSTOM_CHALLENGE`.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the define auth challenge trigger. To pass this data to your Lambda function, you can use the `ClientMetadata` parameter in the AdminRespondToAuthChallenge and

RespondToAuthChallenge API operations. The request that invokes the define auth challenge function doesn't include data passed in the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations.

Define Auth challenge response parameters

In the response, you can return the next stage of the authentication process.

challengeName

A string that contains the name of the next challenge. If you want to present a new challenge to your user, specify the challenge name here.

issueTokens

If you determine that the user has completed the authentication challenges sufficiently, set to `true`. If the user has not met the challenges sufficiently, set to `false`.

failAuthentication

If you want to end the current authentication process, set to `true`. To continue the current authentication process, set to `false`.

Define Auth challenge example

This example defines a series of challenges for authentication and issues tokens only if the user has completed all of the challenges successfully. When users complete SRP authentication with the `SRP_A` and `PASSWORD_VERIFIER` challenges, this function passes them a `CUSTOM_CHALLENGE` that invokes the create auth challenge trigger. In combination with our create auth challenge example, this sequence delivers a CAPTCHA challenge for challenge three and a security question for challenge four.

After the user solves the CAPTCHA and answers the security question, this function confirms that your user pool can issue tokens. SRP authentication isn't required; you can also set the CAPTCHA and security question as challenges one & two. In the case where your define auth challenge function doesn't declare SRP challenges, your users' success is determined entirely by their responses to your custom prompts.

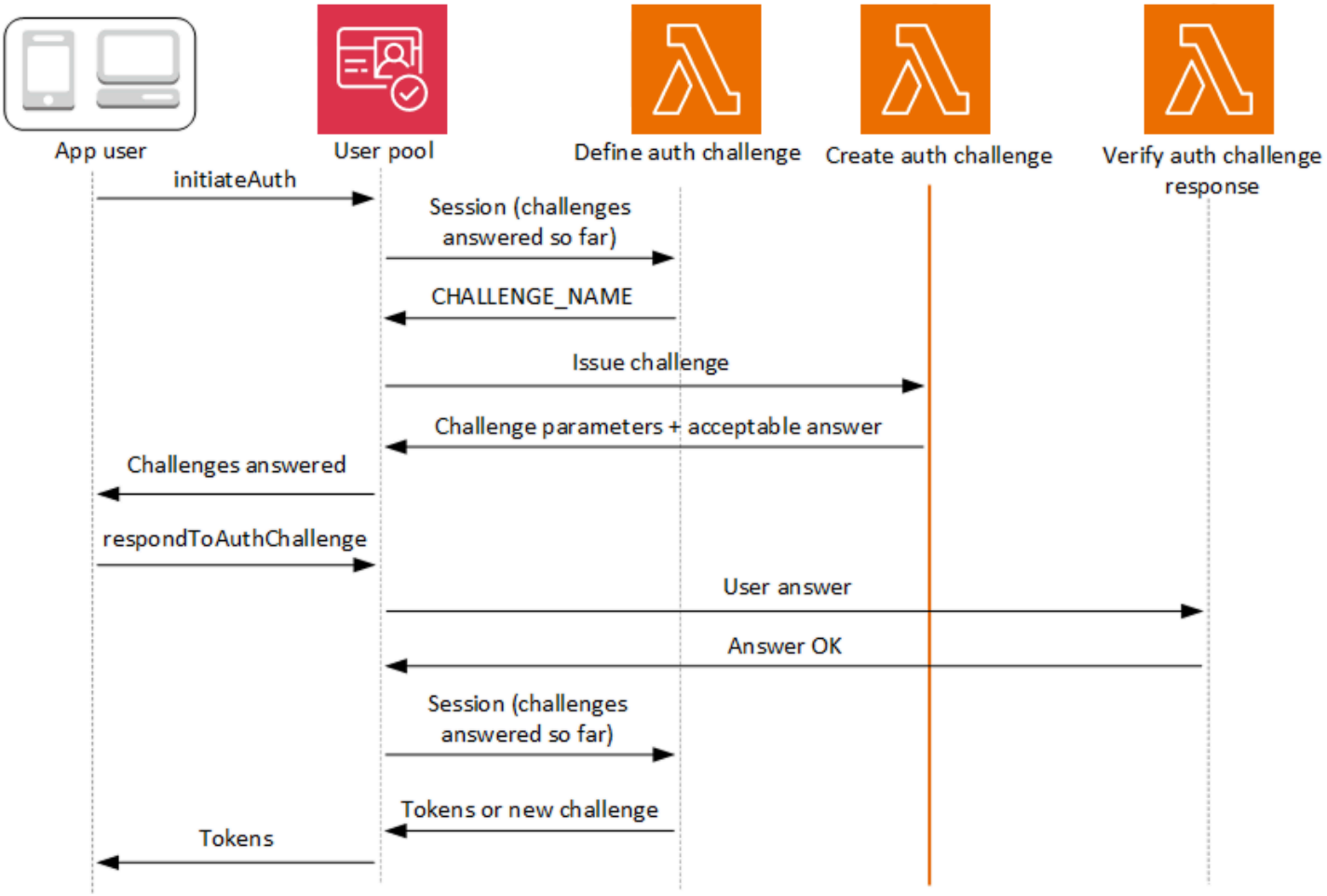
Node.js

```
const handler = async (event) => {
```

```
  if (
    event.request.session.length === 1 &&
    event.request.session[0].challengeName === "SRP_A"
  ) {
    event.response.issueTokens = false;
    event.response.failAuthentication = false;
    event.response.challengeName = "PASSWORD_VERIFIER";
  } else if (
    event.request.session.length === 2 &&
    event.request.session[1].challengeName === "PASSWORD_VERIFIER" &&
    event.request.session[1].challengeResult === true
  ) {
    event.response.issueTokens = false;
    event.response.failAuthentication = false;
    event.response.challengeName = "CUSTOM_CHALLENGE";
  } else if (
    event.request.session.length === 3 &&
    event.request.session[2].challengeName === "CUSTOM_CHALLENGE" &&
    event.request.session[2].challengeResult === true
  ) {
    event.response.issueTokens = false;
    event.response.failAuthentication = false;
    event.response.challengeName = "CUSTOM_CHALLENGE";
  } else if (
    event.request.session.length === 4 &&
    event.request.session[3].challengeName === "CUSTOM_CHALLENGE" &&
    event.request.session[3].challengeResult === true
  ) {
    event.response.issueTokens = true;
    event.response.failAuthentication = false;
  } else {
    event.response.issueTokens = false;
    event.response.failAuthentication = true;
  }

  return event;
};

export { handler };
```

Create Auth challenge Lambda trigger

The create auth challenge trigger is a Lambda function that has the details of each challenge declared by the define auth challenge trigger. It processes the challenge name declared by the define auth challenge trigger and returns a `publicChallengeParameters` that your application must present to the user. This function then provides your user pool with the answer to the challenge, `privateChallengeParameters`, that your user pool passes to the verify auth challenge trigger. Where your define auth challenge trigger manages the challenge sequence, your create auth challenge trigger manages the challenge contents.



Create auth challenge

Amazon Cognito invokes this trigger after **Define Auth Challenge** if a custom challenge has been specified as part of the **Define Auth Challenge** trigger. It creates a [custom authentication flow].

This Lambda trigger is invoked to create a challenge to present to the user. The request for this Lambda trigger includes the `challengeName` and `session`. The `challengeName` is a string and is the name of the next challenge to the user. The value of this attribute is set in the Define Auth Challenge Lambda trigger.

The challenge loop will repeat until all challenges are answered.

Topics

- Create Auth challenge Lambda trigger parameters
- Create Auth challenge example

Create Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "challengeName": "string",
        "session": [
            ChallengeResult,
            . . .
        ],
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "publicChallengeParameters": {
            "string": "string",
            . . .
        },
        "privateChallengeParameters": {
```

```
            "string": "string",
            . . .
        },
        "challengeMetadata": "string"
    }
}
```

Create Auth challenge request parameters

userAttributes

One or more name-value pairs representing user attributes.

userNotFound

This boolean is populated when `PreventUserExistenceErrors` is set to `ENABLED` for your User Pool client.

challengeName

The name of the new challenge.

session

The session element is an array of `ChallengeResult` elements, each of which contains the following elements:

challengeName

The challenge type. One of: `"CUSTOM_CHALLENGE"`, `"PASSWORD_VERIFIER"`, `"SMS_MFA"`, `"DEVICE_SRP_AUTH"`, `"DEVICE_PASSWORD_VERIFIER"`, `"NEW_PASSWORD_REQUIRED"`, or `"ADMIN_NO_SRP_AUTH"`.

challengeResult

Set to `true` if the user successfully completed the challenge, or `false` otherwise.

challengeMetadata

Your name for the custom challenge. Used only if `challengeName` is `"CUSTOM_CHALLENGE"`.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the create auth challenge trigger. You can use the ClientMetadata parameter in

the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions to pass this data to your Lambda function. The request that invokes the create auth challenge function does not include data passed in the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations.

Create Auth challenge response parameters

publicChallengeParameters

One or more key-value pairs for the client app to use in the challenge to be presented to the user. This parameter should contain all of the necessary information to present the challenge to the user accurately.

privateChallengeParameters

This parameter is only used by the Verify Auth Challenge Response Lambda trigger. This parameter should contain all of the information that is required to validate the user's response to the challenge. In other words, the `publicChallengeParameters` parameter contains the question that is presented to the user and `privateChallengeParameters` contains the valid answers for the question.

challengeMetadata

Your name for the custom challenge, if this is a custom challenge.

Create Auth challenge example

This function has two custom challenges that correspond to the challenge sequence in our define auth challenge example. The first two challenges are SRP authentication. For the third challenge, this function returns a CAPTCHA URL to your application in the challenge response. Your application renders the CAPTCHA at the given URL and returns the user's input. The URL for the CAPTCHA image is added to the public challenge parameters as "`captchaUrl`", and the expected answer is added to the private challenge parameters.

For the fourth challenge, this function returns a security question. Your application renders the question and prompts the user for their answer. After users solve both custom challenges, the define auth challenge trigger confirms that your user pool can issue tokens.

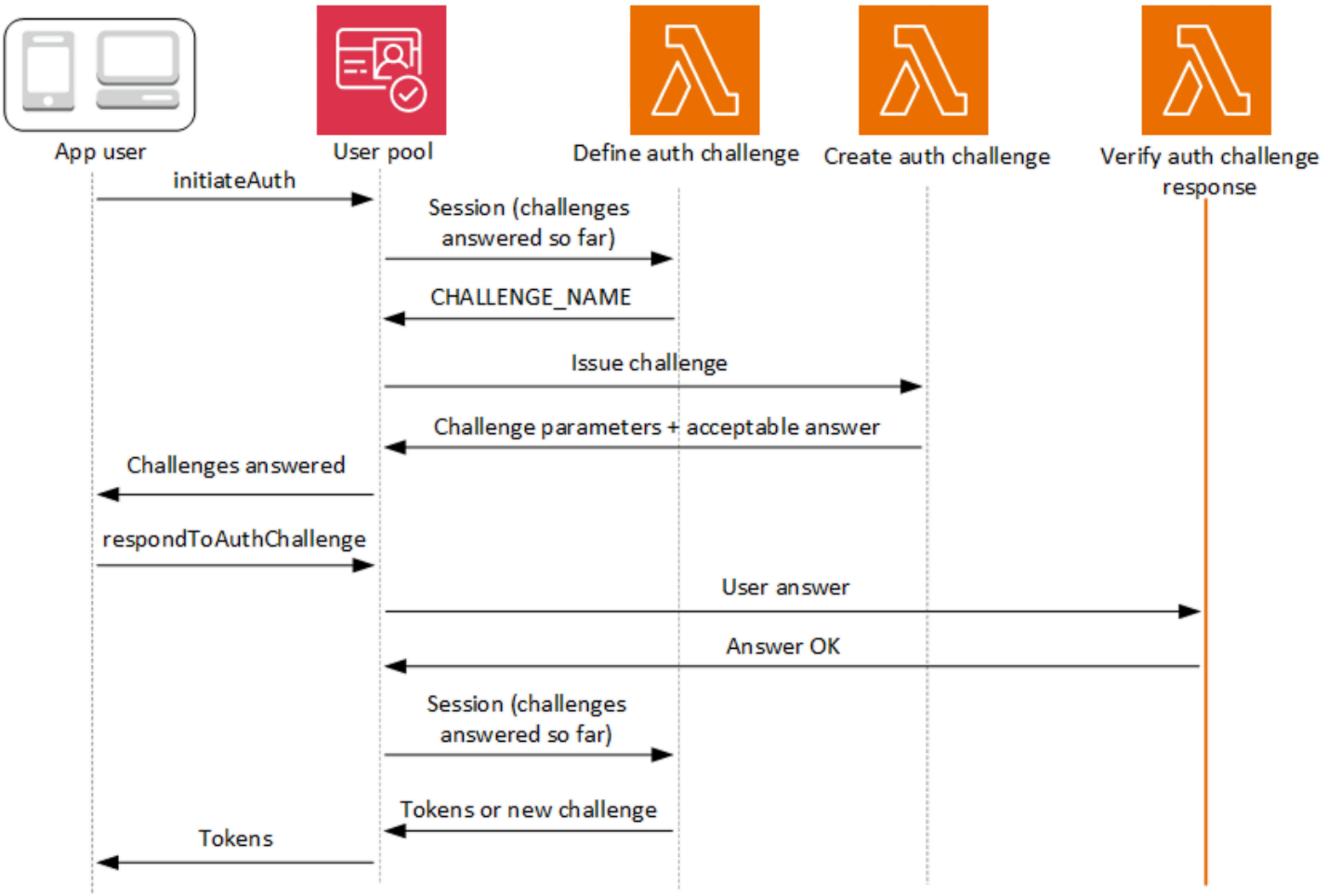
Node.js

```
const handler = async (event) => {
```

```
  if (event.request.challengeName !== "CUSTOM_CHALLENGE") {
    return event;
  }

  if (event.request.session.length === 2) {
    event.response.publicChallengeParameters = {};
    event.response.privateChallengeParameters = {};
    event.response.publicChallengeParameters.captchaUrl = "url/123.jpg";
    event.response.privateChallengeParameters.answer = "5";
  }

  if (event.request.session.length === 3) {
    event.response.publicChallengeParameters = {};
    event.response.privateChallengeParameters = {};
    event.response.publicChallengeParameters.securityQuestion =
      "Who is your favorite team mascot?";
    event.response.privateChallengeParameters.answer = "Peccy";
  }

  return event;
};

export { handler };
```

Verify Auth challenge response Lambda trigger

The verify auth challenge trigger is a Lambda function that compares a user's provided response to a known answer. This function tells your user pool whether the user answered the challenge correctly. When the verify auth challenge trigger responds with an `answerCorrect` of `true`, the authentication sequence can continue.



Verify auth challenge response

Amazon Cognito invokes this trigger to verify if the response from the user for a custom Auth Challenge is valid or not. It is part of a user pool [custom authentication flow](#).

The request for this trigger contains the `privateChallengeParameters` and `challengeAnswer` parameters. The Create Auth Challenge Lambda trigger returns `privateChallengeParameters` values, and contains the expected response from the user. The `challengeAnswer` parameter contains the user's response for the challenge.

The response contains the `answerCorrect` attribute. If the user successfully completes the challenge, Amazon Cognito sets the attribute value to `true`. If the user doesn't successfully complete the challenge, Amazon Cognito sets the value to `false`.

The challenge loop repeats until the users answers all challenges.

Topics

- Verify Auth challenge Lambda trigger parameters
- Verify Auth challenge response example

Verify Auth challenge Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "privateChallengeParameters": {
            "string": "string",
            . . .
        },
        "challengeAnswer": "string",
        "clientMetadata": {
            "string": "string",
            . . .
        },
        "userNotFound": boolean
    },
    "response": {
        "answerCorrect": boolean
    }
}
```

Verify Auth challenge request parameters

userAttributes

This parameter contains one or more name-value pairs that represent user attributes.

userNotFound

When Amazon Cognito sets `PreventUserExistenceErrors` to `ENABLED` for your user pool client, Amazon Cognito populates this Boolean .

privateChallengeParameters

This parameter comes from the Create Auth Challenge trigger. To determine whether the user passed a challenge, Amazon Cognito compares the parameters against a user's **challengeAnswer**.

This parameter contains all of the information that is required to validate the user's response to the challenge. That information includes the question that Amazon Cognito presents to the user (`publicChallengeParameters`), and the valid answers for the question (`privateChallengeParameters`). Only the Verify Auth Challenge Response Lambda trigger uses this parameter.

challengeAnswer

This parameter value is the answer from the user's response to the challenge.

clientMetadata

This parameter contains one or more key-value pairs that you can provide as custom input to the Lambda function for the verify auth challenge trigger. To pass this data to your Lambda function, use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API operations. Amazon Cognito doesn't include data from the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the verify auth challenge function.

Verify Auth challenge response parameters

answerCorrect

If the user successfully completes the challenge, Amazon Cognito sets this parameter to `true`. If the user doesn't successfully complete the challenge, Amazon Cognito sets the parameter to `false`.

Verify Auth challenge response example

This verify auth challenge function checks whether the user's response to a challenge matches the expected response. The user's answer is defined by input from your application and the preferred answer is defined by `privateChallengeParameters.answer` in the response from the create auth challenge trigger response. Both the correct answer and the given answer are part of the input event to this function.

In this example, if the user's response matches the expected response, Amazon Cognito sets the `answerCorrect` parameter to `true`.

Node.js

```
const handler = async (event) => {
  if (
    event.request.privateChallengeParameters.answer ===
    event.request.challengeAnswer
  ) {
    event.response.answerCorrect = true;
  } else {
    event.response.answerCorrect = false;
  }

  return event;
};

export { handler };
```

Pre token generation Lambda trigger

Because Amazon Cognito invokes this trigger before token generation, you can customize the claims in user pool tokens. With the **Basic features** of the version one or `V1_0` pre token generation trigger event, you can customize the identity (ID) token. In user pools with the Essentials or Plus feature plan, you can generate the version two or `V2_0` trigger event with access token customization, and the version three or `V3_0` trigger event with access token customization for machine-to-machine (M2M) authorization, including both client-credentials grants and the `GetClientToken` API operation.

Amazon Cognito sends a `V1_0` event as a request to your function with data that it would write to the ID token. A `V2_0` or `V3_0` event is a single request with the data that Amazon Cognito would write to both the identity and access tokens. To customize both tokens, you must update your function to use trigger version two or three, and send data for both tokens in the same response.

Amazon Cognito applies version two event responses to access tokens from user authentication, where a human user has presented credentials to your user pool. Version three event responses apply to access tokens from user authentication and machine authentication, where automated

systems authorize access token requests with app client secrets. Aside from the circumstances of the resulting access tokens, version two and three events are identical.

This Lambda trigger can add, remove, and modify some claims in identity and access tokens before Amazon Cognito issues them to your app. To use this feature, associate a Lambda function from the Amazon Cognito user pools console or update your user pool `LambdaConfig` through the AWS Command Line Interface (AWS CLI).

Event versions

Your user pool can deliver different versions of a pre token generation trigger event to your Lambda function. A `V1_0` trigger delivers the parameters for modification of ID tokens. A `V2_0` or `V3_0` trigger delivers parameters for the following.

1. The functions of a `V1_0` trigger.
2. The ability to customize access tokens.
3. The ability to pass complex datatypes to ID and access token claim values:
   - String
   - Number
   - Boolean
   - Array of strings, numbers, booleans, or a combination of any of these
   - JSON

> **Note**
>
> In the ID token, you can populate complex objects to the values of claims except for `phone_number_verified`, `email_verified`, `updated_at`, and `address`.

User pools deliver `V1_0` events by default. To configure your user pool to send a `V2_0` event, choose a **Trigger event version** of **Basic features + access token customization for user identities** when you configure your trigger in the Amazon Cognito console. To produce `V3_0` events, choose **Basic features + access token customization for user and machine identities**. You can also set the value of `LambdaVersion` in the LambdaConfig parameters in an UpdateUserPool or CreateUserPool API request. Event versions one, two, and three are available in the **Essentials** and

Plus feature plans. M2M operations for version three events have a pricing structure separate from the monthly active users (MAU) formula. For more information, see Amazon Cognito Pricing.

> **Note**
>
> User pools that were operational with the **Advanced security features** option on or before November 22, 2024 at 1800 GMT, and that remain on the **Lite** feature tier have access to event versions one and two of the pre token generation trigger. User pools in this legacy tier *without* advanced security features have access to event version one. Version three is *only* available in Essentials and Plus.

Claims and scopes reference

Amazon Cognito limits the claims and scopes that you can add, modify, or suppress in access and identity tokens. The following table describes the claims that your Lambda function can and can't modify, and the trigger event parameters that affect the presence or value of the claim.

Claim	Defaul token type	Can add?	Can modify	Can suppre	Event parameter - add or modify	Event parameter - suppress	Identity type	Event version
Any claim not in the user pool token schema	None	Yes	Yes	N/A	`claimsTo/ ddOrOver: ide`	`claimsTo: uppress`	User, machine[1]	All[2]
`scope`	Access	Yes	Yes	Yes	`scopesTo/ dd`	`scopesTo: uppress`	User, machine[1]	`v2_0`, `v3_0`
`cognito:( roups`	ID, Access	Yes	Yes	Yes	`groupsTo( verride`	`claimsTo: uppress`	User	All[2]

Claim	Default token type	Can add?	Can modify	Can suppress	Event parameter - add or modify	Event parameter - suppress	Identity type	Event version
cognito:preferred_role	ID	Yes	Yes	Yes	preferredRole	claimsToSuppress	User	All
cognito:roles	ID	Yes	Yes	Yes	iamRolesToOverride	claimsToSuppress	User	All
cognito:username	ID	No	No	No	N/A	N/A	User	N/A
Any other claim with a cognito: prefix	None	No	No	No	N/A	N/A	N/A	N/A
username	Access	No	No	No	N/A	N/A	User	v2_0, v3_0
sub	ID, Access	No	No	No	N/A	N/A	User	N/A
standard OIDC attribute	ID	Yes	Yes	Yes	claimsToAddOrOverride	claimsToSuppress	User	All
custom: attribute	ID	Yes	Yes	Yes	claimsToAddOrOverride	claimsToSuppress	User	All
dev: attribute	ID	No	No	Yes	N/A	claimsToSuppress	User	All

Claim	Default token type	Can add?	Can modify	Can suppress	Event parameter - add or modify	Event parameter - suppress	Identity type	Event version
identities	ID	No	No	No	N/A	N/A	User	N/A
aud[4]	ID	No	No	No	N/A	N/A	User, machine	N/A
client_id	Access	No	No	No	N/A	N/A	User, machine	N/A
event_id	Access	No	No	No	N/A	N/A	User, machine	N/A
device_key	Access	No	No	No	N/A	N/A	User	N/A
version	Access	No	No	No	N/A	N/A	User, machine	N/A
acr	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
amr	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
at_hash	ID	No	No	No	N/A	N/A	User, machine	N/A
auth_time	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
azp	ID, Access	No	No	No	N/A	N/A	User, machine	N/A

Claim	Default token type	Can add?	Can modify	Can suppre	Event parameter - add or modify	Event parameter - suppress	Identity type	Event version
exp	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
iat	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
iss	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
jti	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
nbf	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
nonce	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
origin_ji	ID, Access	No	No	No	N/A	N/A	User, machine	N/A
token_us	ID, Access	No	No	No	N/A	N/A	User, machine	N/A

[1] Access tokens for machine identities are only available with `v3_0` of the trigger input event. Event version three is only available in the **Essentials** and **Plus** feature tiers. User pools on the **Lite** tier can receive `v1_0` events. User pools on the **Lite** tier with advanced security features can receive `v1_0` and `v2_0` events.

[2] Configure your pre token generation trigger to event version `v1_0` for ID token only, `v2_0` for ID and access token, `v3_0` for ID and access token with capabilities for machine identities.

[3] To suppress the `cognito:preferred_role` and `cognito:roles` claims, add `cognito:groups` to `claimsToSuppress`.

[4] You can add an `aud` claim to access tokens, but its value must match the app client ID of the current session. You can derive the client ID in the request event from `event.callerContext.clientId`.

Customizing the identity token

With all event versions of the pre token generation Lambda trigger, you can customize the content of an identity (ID) token from your user pool. The ID token provides user attributes from a trusted identity source for sign-in to a web or mobile app. For more information about ID tokens, see Understanding the identity (ID) token.

The uses of the pre token generation Lambda trigger with an ID token include the following.

- Make a change at runtime to the IAM role that your user requests from an identity pool.
- Add user attributes from an external source.
- Add or replace existing user attribute values.
- Suppress disclosure of user attributes that, because of your user's authorized scopes and the read access to attributes that you granted to your app client, would otherwise be passed to your app.

Customizing the access token

With event versions two and three of the pre token generation Lambda trigger, you can customize the content of an access token from your user pool. The access token authorizes users to retrieve information from access-protected resources like Amazon Cognito token-authorized API operations and third-party APIs. For machine-to-machine (M2M) authorization, whether the access token comes from a client credentials grant or from the `GetClientToken` API operation, Amazon Cognito only invokes the pre token generation trigger when your user pool is configured for a version three (`V3_0`) event. For more information about access tokens, see Understanding the access token.

The uses of the pre token generation Lambda trigger with an access token include the following.

- Add or suppress scopes in the `scope` claim. For example, you can add scopes to an access token that resulted from Amazon Cognito user pools API authentication, which only assigns the scope `aws.cognito.signin.user.admin`.
- Change a user's membership in user pool groups.
- Add claims that aren't already present in an Amazon Cognito access token.

- Suppress disclosure of claims that would otherwise be passed to your app.

To support access customization in your user pool, you must configure the user pool to generate an updated version of the trigger request. Update your user pool as shown in the following procedure.

AWS Management Console

To support access token customization in a pre token generation Lambda trigger

1. Go to the Amazon Cognito console, and then choose **User Pools**.
2. Choose an existing user pool from the list, or create a user pool.
3. Choose the **Extensions** menu and locate **Lambda triggers**.
4. Add or edit a **Pre token generation trigger**.
5. Choose a Lambda function under **Assign Lambda function**.
6. Choose a **Trigger event version** of **Basic features + access token customization for user identities** or **Basic features + access token customization for user and machine identities**. This setting updates the request parameters that Amazon Cognito sends to your function to include fields for access token customization.

User pools API

To support access token customization in a pre token generation Lambda trigger

Generate a CreateUserPool or UpdateUserPool API request. You must specify a value for all parameters that you don't want set to a default value. For more information, see Updating user pool and app client configuration.

Include the following content in the `LambdaVersion` parameter of your request. A `LambdaVersion` value of `V2_0` causes your user pool to add parameters for, and apply changes to, access tokens. A `LambdaVersion` value of `V3_0` produces the same event as `V2_0`, but causes your user pool to *also* apply changes to M2M access tokens. To invoke a specific function version, use a Lambda function ARN with a function version as the value of `LambdaArn`.

```
"PreTokenGenerationConfig": {
   "LambdaArn": "arn:aws:lambda:us-west-2:123456789012:function:MyFunction",
   "LambdaVersion": "V3_0"
},
```

Client metadata for machine-to-machine (M2M) client credentials

You can pass client metadata in M2M requests. Client metadata is additional information from a user or application environment that can contribute to the outcomes of a Pre token generation Lambda trigger. In authentication operations with a user principal, you can pass client metadata to the pre token generation trigger in the body of AdminRespondToAuthChallenge and RespondToAuthChallenge API requests. Because applications conduct the flow for generation of access tokens for M2M with direct requests to the Token endpoint, they have a different model. In the POST body of token requests for client credentials, pass an `aws_client_metadata` parameter with the client metadata object URL-encoded (`x-www-form-urlencoded`) to string. For an example request, see Client credentials with basic authorization. The following is an example parameter that passes the key-value pairs `{"environment": "dev", "language": "en-US"}`.

```
aws_client_metadata=%7B%22environment%22%3A%20%22dev%22,%20%22language%22%3A%20%22en-US%22%7D
```

More resources

- How to customize access tokens in Amazon Cognito user pools

Topics

- Pre token generation Lambda trigger sources
- Pre token generation Lambda trigger parameters
- Pre token trigger event version two example: Add and suppress claims, scopes, and groups
- Pre token generation event version two example: Add claims with complex objects
- Pre token generation event version one example: Add a new claim and suppress an existing claim
- Pre token generation event version one example: Modify the user's group membership

Pre token generation Lambda trigger sources

triggerSource value	Event
`TokenGeneration_HostedAuth`	Called during authentication from the Amazon Cognito managed login sign-in page.

triggerSource value	Event
TokenGeneration_Authentication	Called after user authentication flows have completed.
TokenGeneration_NewPasswordChallenge	Called after the user is created by an admin. This flow is invoked when the user has to change a temporary password.
TokenGeneration_ClientCredentials	Called after an M2M client credentials grant. Your user pool only sends this event when your event version is V3_0.
TokenGeneration_AuthenticateDevice	Called at the end of the authentication of a user device.
TokenGeneration_RefreshTokens	Called when a user tries to refresh the identity and access tokens.

Pre token generation Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests. When you add a pre token generation Lambda trigger to your user pool, you can choose a trigger version. This version determines whether Amazon Cognito passes a request to your Lambda function with additional parameters for access-token customization.

Version one

The version one token can set group membership, IAM roles, and new claims in ID tokens. Group membership overrides also apply to the `cognito:groups` claim in access tokens.

```
{
    "request": {
        "userAttributes": {"string": "string"},
        "groupConfiguration": {
                "groupsToOverride": [
                    "string",
                    "string"
```

```
                ],
                "iamRolesToOverride": [
                    "string",
                    "string"
                ],
                "preferredRole": "string"
            },
            "clientMetadata": {"string": "string"}
        },
        "response": {
            "claimsOverrideDetails": {
                "claimsToAddOrOverride": {"string": "string"},
                "claimsToSuppress": [
                    "string",
                    "string"
                ],
                "groupOverrideDetails": {
                    "groupsToOverride": [
                        "string",
                        "string"
                    ],
                    "iamRolesToOverride": [
                        "string",
                        "string"
                    ],
                    "preferredRole": "string"
                }
            }
        }
}
```

Versions two and three

The versions two and three request events add fields that customize the access token. User pools apply changes from version three events to access tokens for machine identities. These versions also add support for complex `claimsToOverride` data types in the response object. Your Lambda function can return the following types of data in the value of `claimsToOverride`:

- String
- Number
- Boolean

- Array of strings, numbers, booleans, or a combination of any of these
- JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string"
        },
        "scopes": ["string", "string"],
        "groupConfiguration": {
            "groupsToOverride": ["string", "string"],
            "iamRolesToOverride": ["string", "string"],
            "preferredRole": "string"
        },
        "clientMetadata": {
            "string": "string"
        }
    },
    "response": {
        "claimsAndScopeOverrideDetails": {
            "idTokenGeneration": {
                "claimsToAddOrOverride": {
                    "string": [accepted datatype]
                },
                "claimsToSuppress": ["string", "string"]
            },
            "accessTokenGeneration": {
                "claimsToAddOrOverride": {
                    "string": [accepted datatype]
                },
                "claimsToSuppress": ["string", "string"],
                "scopesToAdd": ["string", "string"],
                "scopesToSuppress": ["string", "string"]
            },
            "groupOverrideDetails": {
                "groupsToOverride": ["string", "string"],
                "iamRolesToOverride": ["string", "string"],
                "preferredRole": "string"
            }
        }
    }
}
```

Pre token generation request parameters

Name	Description	Minimum trigger event version
userAttributes	The attributes of your user's profile in your user pool.	1
groupConfiguration	The input object that contains the current group configuration. The object includes `groupsToOverride`, `iamRolesToOverride`, and `preferredRole`.	1
groupsToOverride	The user pool groups that your user is a member of.	1
iamRolesToOverride	You can associate a user pool group with an AWS Identity and Access Management (IAM) role. This element is a list of all IAM roles from the groups that your user is a member of.	1
preferredRole	You can set a precedence for user pool groups. This element contains the name of the IAM role from the group with the highest precendence in the `groupsToOverride` element.	1
clientMetadata	One or more key-value pairs that you can specify and provide as custom input to the Lambda function for the pre token generation trigger. To pass this data to your Lambda function, use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API operations. Amazon Cognito doesn't include data from the `ClientMetadata` parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the pre token generation function.	1

Name	Description	Minimum trigger event version
scopes	Access token scopes. The scopes that are present in an access token are the user pool standard and custom scopes that your user requested, and that you authorize d your app client to issue.	2

Pre token generation response parameters

Name	Description	Minimum trigger event version
claimsOve rrideDetails	A container for all elements in a `V1_0` trigger event.	1
claimsAnd ScopeOver rideDetails	A container for all elements in a `V2_0` or `V3_0` trigger event.	2
idTokenGe neration	The claims that you want to override, add, or suppress in your user's ID token. This parent to ID token customiza tion values appears only in event version 2 and above, but the child elements appear in version 1 events.	2
accessTok enGeneration	The claims and scopes that you want to override, add, or suppress in your user's access token. This parent to access token customization values appears only in event version 2 and above.	2
claimsToA ddOrOverride	A map of one or more claims and their values that you want to add or modify. For group-related claims, use `groupOverrideDetails` instead.	1*

Name	Description	Minimum trigger event version
	In event version 2 and above, this element appears under both `accessTokenGeneration` and `idTokenGeneration`.	
claimsToSuppress	A list of claims that you want Amazon Cognito to suppress. If your function both suppresses and replaces a claim value, then Amazon Cognito suppresses the claim. In event version 2 and above, this element appears under both `accessTokenGeneration` and `idTokenGeneration`.	1
groupOverrideDetails	The output object that contains the current group configuration. The object includes `groupsToOverride`, `iamRolesToOverride`, and `preferredRole`. Your function replaces the `groupOverrideDetails` object with the object that you provide. If you provide an empty or null object in the response, then Amazon Cognito suppresses the groups. To keep the existing group configuration the same, copy the value of the `groupConfiguration` object of the request to the `groupOverrideDetails` object in the response. Then pass it back to the service. Amazon Cognito ID and access tokens both contain the `cognito:groups` claim. Your `groupOverrideDetails` object replaces the `cognito:groups` claim in access tokens and ID tokens. Group overrides are the only changes to the access token that version 1 events can make.	1

Name	Description	Minimum trigger event version
scopesToAdd	A list of scopes that you want to add to the `scope` claim in your user's access token. You can't add scope values that contain one or more blank-space characters.	2
scopesToSuppress	A list of scopes that you want to remove from the `scope` claim in your user's access token.	2

* Response objects to version one events can return strings. Response objects to version two and three events can return complex objects.

Pre token trigger event version two example: Add and suppress claims, scopes, and groups

This example makes the following modifications to a user's tokens.

1. Sets their `family_name` as `Doe` in the ID token.
2. Prevents `email` and `phone_number` claims from appearing in the ID token.
3. Sets their ID token `cognito:roles` claim to `"arn:aws:iam::123456789012:role\/sns_callerA","arn:aws:iam::123456789012:role\/sns_callerC","arn:aws:iam::123456789012:role\/sns_callerB"`.
4. Sets their ID token `cognito:preferred_role` claim to `arn:aws:iam::123456789012:role/sns_caller`.
5. Adds the scopes `openid`, `email`, and `solar-system-data/asteroids.add` to the access token.
6. Suppresses the scope `phone_number` and `aws.cognito.signin.user.admin` from the access token. Removal of `phone_number` prevents retrieval of the user's phone number from `userInfo`. Removal of `aws.cognito.signin.user.admin` prevents API requests by the user to read and modify their own profile with the Amazon Cognito user pools API.

> **Note**
>
> The removal of `phone_number` from scopes only prevents retrieval of a user's phone number if the remaining scopes in the access token include `openid` and at least one more standard scope. For more information, see About scopes.

7. Sets their ID and access token `cognito:groups` claim to `"new-group-A","new-group-B","new-group-C"`.

JavaScript

```
export const handler = function(event, context) {
  event.response = {
    "claimsAndScopeOverrideDetails": {
      "idTokenGeneration": {
        "claimsToAddOrOverride": {
          "family_name": "Doe"
        },
        "claimsToSuppress": [
          "email",
          "phone_number"
        ]
      },
      "accessTokenGeneration": {
        "scopesToAdd": [
          "openid",
          "email",
          "solar-system-data/asteroids.add"
        ],
        "scopesToSuppress": [
          "phone_number",
          "aws.cognito.signin.user.admin"
        ]
      },
      "groupOverrideDetails": {
        "groupsToOverride": [
          "new-group-A",
          "new-group-B",
          "new-group-C"
        ],
        "iamRolesToOverride": [
```

```
          "arn:aws:iam::123456789012:role/new_roleA",
          "arn:aws:iam::123456789012:role/new_roleB",
          "arn:aws:iam::123456789012:role/new_roleC"
        ],
        "preferredRole": "arn:aws:iam::123456789012:role/new_role"
      }
    }
  };
  // Return to Amazon Cognito
  context.done(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "version": "2",
    "triggerSource": "TokenGeneration_Authentication",
    "region": "us-east-1",
    "userPoolId": "us-east-1_EXAMPLE",
    "userName": "JaneDoe",
    "callerContext": {
        "awsSdkVersion": "aws-sdk-unknown-unknown",
        "clientId": "1example23456789"
    },
    "request": {
        "userAttributes": {
            "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "cognito:user_status": "CONFIRMED",
            "email_verified": "true",
            "phone_number_verified": "true",
            "phone_number": "+12065551212",
            "family_name": "Zoe",
            "email": "Jane.Doe@example.com"
        },
        "groupConfiguration": {
            "groupsToOverride": ["group-1", "group-2", "group-3"],
```

```
            "iamRolesToOverride": ["arn:aws:iam::123456789012:role/sns_caller1",
 "arn:aws:iam::123456789012:role/sns_caller2", "arn:aws:iam::123456789012:role/
sns_caller3"],
            "preferredRole": ["arn:aws:iam::123456789012:role/sns_caller"]
        },
        "scopes": [
            "aws.cognito.signin.user.admin", "openid", "email", "phone"
        ]
    },
    "response": {
        "claimsAndScopeOverrideDetails": []
    }
}
```

Pre token generation event version two example: Add claims with complex objects

This example makes the following modifications to a user's tokens.

1. Adds claims of number, string, boolean, and JSON types to the ID token. This is the only change that version two trigger events makes available to the ID token.
2. Adds claims of number, string, boolean, and JSON types to the access token.
3. Adds three scopes to the access token.
4. Suppresses the `email` claim in the ID and access tokens.
5. Suppresses the `aws.cognito.signin.user.admin` scope in the access token.

JavaScript

```
export const handler = function(event, context) {

    var scopes = ["MyAPI.read", "MyAPI.write", "MyAPI.admin"]
    var claims = {}
    claims["aud"]= event.callerContext.clientId;
    claims["booleanTest"] = false;
    claims["longTest"] = 9223372036854775807;
    claims["exponentTest"] = 1.7976931348623157E308;
    claims["ArrayTest"] = ["test", 9223372036854775807, 1.7976931348623157E308,
 true];
    claims["longStringTest"] = "\{\
```

```
        \"first_json_block\": \{\
            \"key_A\": \"value_A\",\
            \"key_B\": \"value_B\"\
        \},\
        \"second_json_block\": \{\
            \"key_C\": \{\
                \"subkey_D\": [\
                    \"value_D\",\
                    \"value_E\"\
                ],\
                \"subkey_F\": \"value_F\"\
            \},\
            \"key_G\": \"value_G\"\
        \}\
    \}";
    claims["jsonTest"] = {
     "first_json_block": {
      "key_A": "value_A",
      "key_B": "value_B"
     },
     "second_json_block": {
      "key_C": {
       "subkey_D": [
        "value_D",
        "value_E"
       ],
       "subkey_F": "value_F"
      },
      "key_G": "value_G"
     }
    };
    event.response = {
        "claimsAndScopeOverrideDetails": {
            "idTokenGeneration": {
                "claimsToAddOrOverride": claims,
                "claimsToSuppress": ["email"]
            },
            "accessTokenGeneration": {
                "claimsToAddOrOverride": claims,
                "claimsToSuppress": ["email"],
                "scopesToAdd": scopes,
                "scopesToSuppress": ["aws.cognito.signin.user.admin"]
            }
        }
```

```
    };
    console.info("EVENT response\n" + JSON.stringify(event, (_, v) => typeof v ===
 'bigint' ? v.toString() : v, 2))
    console.info("EVENT response size\n" + JSON.stringify(event, (_, v) => typeof v
 === 'bigint' ? v.toString() : v).length)
    // Return to Amazon Cognito
    context.done(null, event);
};
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
    "version": "2",
    "triggerSource": "TokenGeneration_HostedAuth",
    "region": "us-west-2",
    "userPoolId": "us-west-2_EXAMPLE",
    "userName": "JaneDoe",
    "callerContext": {
        "awsSdkVersion": "aws-sdk-unknown-unknown",
        "clientId": "1example23456789"
    },
    "request": {
        "userAttributes": {
            "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "cognito:user_status": "CONFIRMED",
            "email_verified": "true",
            "phone_number_verified": "true",
            "phone_number": "+12065551212",
            "email": "Jane.Doe@example.com"
        },
        "groupConfiguration": {
            "groupsToOverride": ["group-1", "group-2", "group-3"],
            "iamRolesToOverride": ["arn:aws:iam::123456789012:role/sns_caller1"],
            "preferredRole": ["arn:aws:iam::123456789012:role/sns_caller1"]
        },
        "scopes": [
            "aws.cognito.signin.user.admin",
```

```
            "phone",
            "openid",
            "profile",
            "email"
        ]
    },
    "response": {
        "claimsAndScopeOverrideDetails": []
    }
}
```

Pre token generation event version one example: Add a new claim and suppress an existing claim

This example uses a version 1 trigger event with a pre token generation Lambda function to add a new claim and suppresses an existing claim.

Node.js

```
const handler = async (event) => {
  event.response = {
    claimsOverrideDetails: {
      claimsToAddOrOverride: {
        my_first_attribute: "first_value",
        my_second_attribute: "second_value",
      },
      claimsToSuppress: ["email"],
    },
  };

  return event;
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample: Because the code example doesn't process any request parameters,

you can use a test event with an empty request. For more information about common request parameters, see User pool Lambda trigger event.

JSON

```
{
  "request": {},
  "response": {}
}
```

Pre token generation event version one example: Modify the user's group membership

This example uses a version 1 trigger event with a pre token generation Lambda function to modify the user's group membership.

Node.js

```
const handler = async (event) => {
  event.response = {
    claimsOverrideDetails: {
      groupOverrideDetails: {
        groupsToOverride: ["group-A", "group-B", "group-C"],
        iamRolesToOverride: [
          "arn:aws:iam::XXXXXXXXXXX:role/sns_callerA",
          "arn:aws:iam::XXXXXXXXX:role/sns_callerB",
          "arn:aws:iam::XXXXXXXXXX:role/sns_callerC",
        ],
        preferredRole: "arn:aws:iam::XXXXXXXXXXX:role/sns_caller",
      },
    },
  };

  return event;
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console,

you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "request": {},
  "response": {}
}
```

Migrate user Lambda trigger

When a user doesn't exist in the user pool at sign-in with a password, or in the forgot-password flow, Amazon Cognito invokes this trigger. After the Lambda function returns successfully, Amazon Cognito creates the user in the user pool. For details on the authentication flow with the user migration Lambda trigger, see Importing users with a user migration Lambda trigger.

To migrate users from your existing user directory into Amazon Cognito user pools at sign-in, or during the forgot-password flow, use this Lambda trigger.

Topics

- Migrate user Lambda trigger sources
- Migrate user Lambda trigger parameters
- Example: Migrate a user with an existing password
- Troubleshooting the migrate user trigger

Migrate user Lambda trigger sources

triggerSource value	Event
`UserMigration_Authentication` [1]	User migration at sign-in.
`UserMigration_ForgotPassword`	User migration during forgot-password flow.

[1] Amazon Cognito doesn't invoke this trigger when users authenticate with passwordless sign-in.

Migrate user Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "userName": "string",
    "request": {
        "password": "string",
        "validationData": {
            "string": "string",
            . . .
        },
        "clientMetadata": {
            "string": "string",
       . . .
        }
    },
    "response": {
        "userAttributes": {
            "string": "string",
            . . .
        },
        "finalUserStatus": "string",
        "messageAction": "string",
        "desiredDeliveryMediums": [ "string", . . .],
        "forceAliasCreation": boolean,
        "enableSMSMFA": boolean
    }
}
```

Migrate user request parameters

userName

The username that the user enters at sign-in.

password

The password that the user enters at sign-in. Amazon Cognito doesn't send this value in a request that's initiated by a forgot-password flow.

validationData

One or more key-value pairs that contain the validation data in the user's sign-in request. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the InitiateAuth and AdminInitiateAuth API actions.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function for the migrate user trigger. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the AdminRespondToAuthChallenge and ForgotPassword API actions.

Migrate user response parameters

userAttributes

This field is required.

This field must contain one or more name-value pairs that Amazon Cognito stores in the user profile in your user pool and uses as user attributes. You can include both standard and custom user attributes. Custom attributes require the `custom:` prefix to distinguish them from standard attributes. For more information, see Custom attributes.

> **Note**
>
> To reset their passwords in the forgot-password flow, a user must have either a verified email address or a verified phone number. Amazon Cognito sends a message containing a reset password code to the email address or phone number in the user attributes.

Attributes	Requirement
Any attributes marked as required when you created your user pool	If any required attributes are missing during the migration, Amazon Cognito uses default values.
`username`	Required if you configured your user pool with alias attributes in addition to username for sign-in, and the user has entered an

Attributes	Requirement
	valid alias value as a username. This alias value can be an email address, preferred username, or phone number. If the request and the user pool meet the alias requirements, the response from your function must assign the `username` parameter that it received to an alias attribute, Also, the response must assign your own value to the `username` attribute. If your user pool doesn't meet the conditions required to map the received `username` to an alias, then the `username` parameter in the response must either exactly match the request, or be omitted. **Note** `username` must be unique in the user pool.

finalUserStatus

You can set this parameter to `CONFIRMED` to auto-confirm your users so that they can sign in with their previous passwords. When you set a user to `CONFIRMED`, they do not need to take additional action before they can sign in. If you don't set this attribute to `CONFIRMED`, it's set to `RESET_REQUIRED`.

A `finalUserStatus` of `RESET_REQUIRED` means that the user must change their password immediately after migration at sign-in, and your client app must handle the `PasswordResetRequiredException` during the authentication flow.

Note

Amazon Cognito doesn't enforce the password strength policy that you configured for the user pool during migration using Lambda trigger. If the password doesn't meet the password policy that you configured, Amazon Cognito still accepts the password so that it can continue to migrate the user. To enforce password strength policy and reject passwords that don't meet the policy, validate the password strength in your code. Then, if the password doesn't meet the policy, set finalUserStatus to `RESET_REQUIRED`.

messageAction

You can set this parameter to SUPPRESS to decline to send the welcome message that Amazon Cognito usually sends to new users. If your function doesn't return this parameter, Amazon Cognito sends the welcome message.

desiredDeliveryMediums

You can set this parameter to EMAIL to send the welcome message by email, or SMS to send the welcome message by SMS. If your function doesn't return this parameter, Amazon Cognito sends the welcome message by SMS.

forceAliasCreation

If you set this parameter to TRUE and the phone number or email address in the UserAttributes parameter already exists as an alias with a different user, the API call migrates the alias from the previous user to the newly created user. The previous user can no longer log in using that alias.

If you set this parameter to FALSE and the alias exists, Amazon Cognito doesn't migrate the user and returns an error to the client app.

If you don't return this parameter, Amazon Cognito assumes its value is "false".

enableSMSMFA

Set this parameter to `true` to require that your migrated user complete SMS text message multi-factor authentication (MFA) to sign in. Your user pool must have MFA enabled. Your user's attributes in the request parameters must include a phone number, or else the migration of that user will fail.

Example: Migrate a user with an existing password

This example Lambda function migrates the user with an existing password and suppresses the welcome message from Amazon Cognito.

Node.js

```
exports.handler = (event, context, callback) => {
  var user;

  if (event.triggerSource == "UserMigration_Authentication") {
    // authenticate the user with your existing user directory service
    user = authenticateUser(event.userName, event.request.password);
```

```
    if (user) {
      event.response.userAttributes = {
        email: user.emailAddress,
        email_verified: "true",
      };
      event.response.finalUserStatus = "CONFIRMED";
      event.response.messageAction = "SUPPRESS";
      context.succeed(event);
    } else {
      // Return error to Amazon Cognito
      callback("Bad password");
    }
  } else if (event.triggerSource == "UserMigration_ForgotPassword") {
    // Lookup the user in your existing user directory service
    user = lookupUser(event.userName);
    if (user) {
      event.response.userAttributes = {
        email: user.emailAddress,
        // required to enable password-reset code to be sent to user
        email_verified: "true",
      };
      event.response.messageAction = "SUPPRESS";
      context.succeed(event);
    } else {
      // Return error to Amazon Cognito
      callback("Bad password");
    }
  } else {
    // Return error to Amazon Cognito
    callback("Bad triggerSource " + event.triggerSource);
  }
};
```

Troubleshooting the migrate user trigger

If migration doesn't work as expected even though your function returns the documented response, check the following common causes.

The trigger isn't invoked

Amazon Cognito invokes the migrate user trigger only when the user doesn't already exist in the user pool. It applies only to the `USER_PASSWORD_AUTH`, `ADMIN_USER_PASSWORD_AUTH`,

and USER_AUTH (with a password) sign-in flows, and to the forgot-password flow. It isn't invoked for SRP (USER_SRP_AUTH) sign-in, because the user's password never reaches your function in that flow, or for passwordless sign-in. If your trigger doesn't fire, confirm the app client permits one of these flows and that the user isn't already present in the pool.

Amazon Cognito can't invoke the function

If you set the trigger with a resource-based policy on the function rather than through the Amazon Cognito console, the policy must allow the `cognito-idp.amazonaws.com` service principal to call `lambda:InvokeFunction`, scoped to your user pool ARN. The console adds this permission for you; a function wired up with infrastructure as code or the Lambda API needs it added explicitly. Without it, migration fails because Amazon Cognito can't invoke the trigger.

The user signs in but no profile is created

Amazon Cognito creates the migrated user only when your function returns the response in the exact shape it expects: the `userAttributes` field must be present and must include every attribute marked required on the user pool. A function that authenticates the user but returns an empty or malformed `response` object, or omits a required attribute, completes the sign-in call without persisting a profile. Return `userAttributes` with all required attributes, and set `finalUserStatus` and `messageAction` as needed.

Custom message Lambda trigger

When you have an external standard for the email and SMS messages that you want to send to your users, or when you want to apply your own logic at runtime to the formatting of user messages, add a custom message trigger to your user pool. The custom message Lambda receives the contents of all email and SMS messages before your user pool sends them. Your Lambda function then has the opportunity to modify the message contents and subject.

Amazon Cognito invokes this trigger before it sends an email or phone verification message or a multi-factor authentication (MFA) code. You can customize the message dynamically with your custom message trigger.

The request includes `codeParameter`. This is a string that acts as a placeholder for the code that Amazon Cognito delivers to the user. Insert the `codeParameter` string into the message body where you want the verification code to appear. When Amazon Cognito receives this response, Amazon Cognito replaces the `codeParameter` string with the actual verification code.

Note

The input event for a custom message Lambda function with the `CustomMessage_AdminCreateUser` trigger source includes a username and verification code. Because an admin-created user must receive both their user name and code, the response from your function must include placeholder variables for the username and code. The placeholders for your message are the values of `request.usernameParameter` and `request.codeParameter`. These values are typically `{username}` and `{####}`; as a best practice, reference the input values instead of hardcoding the variable names.

Note

Messages that you configure in the **Message templates** menu of the Amazon Cognito console can override a body that your custom message function sets. If your function's message doesn't appear, check whether a console message template is configured for the same event. For more information, see Configuring MFA, authentication, verification and invitation messages.

Topics

- Custom message Lambda trigger sources
- Custom message Lambda trigger parameters
- Custom message for sign-up example
- Custom message for admin create user example

Custom message Lambda trigger sources

triggerSource value	Event
`CustomMessage_SignUp`	Custom message – To send the confirmation code post sign-up.
`CustomMessage_AdminCreateUser`	Custom message – To send the temporary password to a new user.

triggerSource value	Event
CustomMessage_ResendCode	Custom message – To resend the confirmation code to an existing user.
CustomMessage_ForgotPassword	Custom message – To send the confirmation code for Forgot Password request.
CustomMessage_UpdateUserAttribute	Custom message – When a user's email or phone number is changed, this trigger sends a verification code automatically to the user. Cannot be used for other attributes.
CustomMessage_VerifyUserAttribute	Custom message – This trigger sends a verification code to the user when they manually request it for a new email or phone number.
CustomMessage_Authentication	Custom message – To send MFA code during authentication.

Custom message Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "userAttributes": {
            "string": "string",
            . . .
        }
        "codeParameter": "####",
        "usernameParameter": "string",
        "clientMetadata": {
            "string": "string",
            . . .
        }
```

```
    },
    "response": {
        "smsMessage": "string",
        "emailMessage": "string",
        "emailSubject": "string"
    }
}
```

Custom message request parameters

userAttributes

One or more name-value pairs representing user attributes.

codeParameter

A string for you to use as the placeholder for the verification code in the custom message.

usernameParameter

The user name. Amazon Cognito includes this parameter in requests that result from admin-created users.

clientMetadata

One or more key-value pairs that you can provide as custom input to the Lambda function that you specify for the custom message trigger. The request that invokes a custom message function doesn't include data passed in the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the following API actions:

- AdminResetUserPassword
- AdminRespondToAuthChallenge
- AdminUpdateUserAttributes
- ForgotPassword
- GetUserAttributeVerificationCode
- ResendConfirmationCode
- SignUp
- UpdateUserAttributes

Custom message response parameters

In the response, specify the custom text to use in messages to your users. For the string constraints that Amazon Cognito applies to these parameters, see MessageTemplateType.

smsMessage

The custom SMS message to be sent to your users. Must include the `codeParameter` value that you received in the request.

emailMessage

The custom email message to send to your users. You can use HTML formatting in the `emailMessage` parameter. Must include the `codeParameter` value that you received in the request as the variable `{####}`. Amazon Cognito can use the `emailMessage` parameter only if the `EmailSendingAccount` attribute of the user pool is `DEVELOPER`. If the `EmailSendingAccount` attribute of the user pool isn't `DEVELOPER` and an `emailMessage` parameter is returned, Amazon Cognito generates a 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException`. When you choose Amazon Simple Email Service (Amazon SES) to send email messages, the `EmailSendingAccount` attribute of a user pool is `DEVELOPER`. Otherwise, the value is `COGNITO_DEFAULT`.

emailSubject

The subject line for the custom message. You can only use the `emailSubject` parameter if the EmailSendingAccount attribute of the user pool is `DEVELOPER`. If the `EmailSendingAccount` attribute of the user pool isn't `DEVELOPER` and Amazon Cognito returns an `emailSubject` parameter, Amazon Cognito generates a 400 error code `com.amazonaws.cognito.identity.idp.model.InvalidLambdaResponseException`. The `EmailSendingAccount` attribute of a user pool is `DEVELOPER` when you choose to use Amazon Simple Email Service (Amazon SES) to send email messages. Otherwise, the value is `COGNITO_DEFAULT`.

Custom message for sign-up example

This example Lambda function customizes an email or SMS message when the service requires an app to send a verification code to the user.

Amazon Cognito can invoke a Lambda trigger at multiple events: post-registration, resending a verification code, recovering a forgotten password, or verifying a user attribute. The response

includes messages for both SMS and email. The message must include the code parameter "####". This parameter is the placeholder for the verification code that the user receives.

The maximum length for an email message is 20,000 UTF-8 characters. This length includes the verification code. You can use HTML tags in these email messages.

The maximum length of SMS messages is 140 UTF-8 characters. This length includes the verification code.

> **Note**
>
> If your function returns an email message longer than 20,000 UTF-8 characters, Amazon Cognito doesn't return an error. Instead, it silently discards your custom message and sends the default message for that event. If your custom message isn't reaching users, check its length against this limit first.

Node.js

```
const handler = async (event) => {
  if (event.triggerSource === "CustomMessage_SignUp") {
    const message = `Thank you for signing up. Your confirmation code is
 ${event.request.codeParameter}.`;
    event.response.smsMessage = message;
    event.response.emailMessage = message;
    event.response.emailSubject = "Welcome to the service.";
  }
  return event;
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
```

```
 "version": "1",
 "region": "us-west-2",
 "userPoolId": "us-west-2_EXAMPLE",
 "userName": "test-user",
 "callerContext": {
  "awsSdkVersion": "aws-sdk-unknown-unknown",
  "clientId": "1example23456789"
 },
 "triggerSource": "CustomMessage_SignUp",
 "request": {
  "userAttributes": {
   "sub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
   "cognito:user_status": "CONFIRMED",
   "email_verified": "true",
   "phone_number_verified": "true",
   "phone_number": "+12065551212",
   "email": "test-user@example.com"
  },
  "codeParameter": "{####}",
  "linkParameter": "{##Click Here##}",
  "usernameParameter": "None"
 },
 "response": {
  "smsMessage": "None",
  "emailMessage": "None",
  "emailSubject": "None"
 }
}
```

Custom message for admin create user example

The request that Amazon Cognito sent to this example custom message Lambda function has a `triggerSource` value of `CustomMessage_AdminCreateUser` and a username and temporary password. The function populates `${event.request.codeParameter}` from the temporary password in the request, and `${event.request.usernameParameter}` from the username in the request.

Your custom messages must insert the values of `codeParameter` and `usernameParameter` into `smsMessage` and `emailMessage` in the response object. In this example, the function writes the same message to the response fields `event.response.smsMessage` and `event.response.emailMessage`.

The maximum length of an email message is 20,000 UTF-8 characters. This length includes the verification code. You can use HTML tags in these emails. The maximum length of SMS messages is 140 UTF-8 characters. This length includes the verification code.

> **Note**
>
> If your function returns an email message longer than 20,000 UTF-8 characters, Amazon Cognito doesn't return an error. Instead, it silently discards your custom message and sends the default message for that event. If your custom message isn't reaching users, check its length against this limit first.

The response includes messages for both SMS and email.

Node.js

```
const handler = async (event) => {
  if (event.triggerSource === "CustomMessage_AdminCreateUser") {
    const message = `Welcome to the service. Your user name is
 ${event.request.usernameParameter}. Your temporary password is
 ${event.request.codeParameter}`;
    event.response.smsMessage = message;
    event.response.emailMessage = message;
    event.response.emailSubject = "Welcome to the service";
  }
  return event;
};

export { handler };
```

Amazon Cognito passes event information to your Lambda function. The function then returns the same event object to Amazon Cognito, with any changes in the response. In the Lambda console, you can set up a test event with data that is relevant to your Lambda trigger. The following is a test event for this code sample:

JSON

```
{
  "version": 1,
  "triggerSource": "CustomMessage_AdminCreateUser",
```

```
  "region": "<region>",
  "userPoolId": "<userPoolId>",
  "userName": "<userName>",
  "callerContext": {
      "awsSdk": "<calling aws sdk with version>",
      "clientId": "<apps client id>",
      ...
  },
  "request": {
      "userAttributes": {
          "phone_number_verified": false,
          "email_verified": true,
           ...
      },
      "codeParameter": "####",
      "usernameParameter": "username"
  },
  "response": {
      "smsMessage": "<custom message to be sent in the message with code parameter
 and username parameter>"
      "emailMessage": "<custom message to be sent in the message with code parameter
 and username parameter>"
      "emailSubject": "<custom email subject>"
  }
}
```

Custom sender Lambda triggers

The Lambda triggers `CustomEmailSender` and `CustomSMSSender` support third-party email and SMS notifications in user pools. You can choose SMS and email providers to send notifications to users from within your Lambda function code. When Amazon Cognito sends invitations, MFA codes, confirmation codes, verification codes, and temporary passwords to users, the events activate your configured Lambda functions. Amazon Cognito sends the code and temporary passwords (secrets) to your activated Lambda functions. Amazon Cognito encrypts these secrets with an AWS KMS customer managed key and the AWS Encryption SDK. The AWS Encryption SDK is a client-side encryption library that helps you to encrypt and decrypt generic data.

CustomEmailSender

Amazon Cognito invokes this trigger to send email notifications to users.

CustomSMSSender

Amazon Cognito invokes this trigger to send SMS notifications to users.

Encryption concepts

Amazon Cognito doesn't send users' codes in plaintext in the events that it sends to custom sender triggers. The Lambda functions must decrypt codes in the events. The following concepts are the encryption architecture that your function must use to get codes that it can deliver to users.

AWS KMS

AWS KMS is a managed service to create and control AWS KMS keys. These keys encrypt your data. For more information see, What is AWS Key Management Service?.

KMS key

A KMS key is a logical representation of a cryptographic key. The KMS key includes metadata, such as the key ID, creation date, description, and key state. The KMS key also contains the key material used to encrypt and decrypt data. For more information see, AWS KMS keys.

Symmetric KMS key

A symmetric KMS key is a 256-bit encryption key that doesn't exit AWS KMS unencrypted. To use a symmetric KMS key, you must call AWS KMS. Amazon Cognito uses symmetric keys. The same key encrypts and decrypts. For more information see, Symmetric KMS keys.

Things to know about custom sender Lambda triggers

- To configure your user pools to use these Lambda triggers, you can use the AWS CLI or SDK. These configurations aren't available from Amazon Cognito console.

  The `UpdateUserPool` operation sets Lambda configuration. Requests to this operation require all the parameters of your user pool *and* the parameters that you want to change. If you don't provide all relevant parameters, Amazon Cognito sets the values of any missing parameters to their defaults. As demonstrated in the AWS CLI example that follows, include entries for all Lambda functions that you want to add to or keep in your user pool. For more information, see Updating user pool and app client configuration.

```
   #Send this parameter in an 'aws cognito-idp update-user-pool' CLI command,
including any existing
   #user pool configurations. This snippet also includes a pre sign-up trigger for
syntax reference. The pre sign-up trigger
   #doesn't have a role in custom sender triggers.

     --lambda-config "PreSignUp=lambda-arn, \
                      CustomSMSSender={LambdaVersion=V1_0,LambdaArn=lambda-arn}, \
                      CustomEmailSender={LambdaVersion=V1_0,LambdaArn=lambda-arn}, \
                      KMSKeyID=key-id"
```

For requests that use the JSON body of `UpdateUserPool` the following `LambdaConfig` snippet assigns custom SMS and email sender functions.

```
"LambdaConfig": {
   "KMSKeyID": "arn:aws:kms:us-
east-1:111122223333:key/a6c4f8e2-0c45-47db-925f-87854bc9e357",
   "CustomEmailSender": {
      "LambdaArn": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
      "LambdaVersion": "V1_0"
   },
   "CustomSMSSender": {
      "LambdaArn": "arn:aws:lambda:us-east-1:111122223333:function:MyFunction",
      "LambdaVersion": "V1_0"
   }
```

- To remove a custom sender Lambda trigger with an `update-user-pool` AWS CLI command, omit the `CustomSMSSender` or `CustomEmailSender` parameter from `--lambda-config`, and include all other triggers that you want to use with your user pool.

  To remove a custom sender Lambda trigger with an `UpdateUserPool` API request, omit the `CustomSMSSender` or `CustomEmailSender` parameter from the request body that contains the rest of your user pool configuration.

- Amazon Cognito HTML-escapes reserved characters like < (`<`) and > (`>`) in your user's temporary password. These characters might appear in temporary passwords that Amazon Cognito sends to your custom email sender function, but don't appear in temporary verification codes. To send temporary passwords, your Lambda function must unescape these characters after it decrypts the password, and before it sends the message to your user.

Activating custom sender Lambda triggers

To use custom logic to send SMS or email messages for your user pool, set up custom sender triggers. The following procedure assigns a custom SMS trigger, a custom email trigger, or both to your user pool. After you add your custom sender trigger, Amazon Cognito always sends user attributes, including the phone number, and the one-time code to your Lambda function instead of the default behavior that sends an SMS or email message.

1. Create a symmetric encryption key in AWS Key Management Service (AWS KMS). Amazon Cognito generates secrets—temporary passwords, verification codes, authentication one-time passwords, and confirmation codes—then uses this KMS key to encrypt the secrets with AWS Encryption SDK. You can then use the AWS Encryption SDK in your Lambda function to decrypt the secrets and send them to the user in plaintext.
2. The IAM principal that creates or updates your user pool creates a one-time grant against the KMS key that Amazon Cognito uses to encrypt the code. Grant this principal `CreateGrant` permissions for your KMS key. For this example KMS key policy to be effective, the administrator who updates the user pool must be signed in with an assumed-role session for the IAM role `arn:aws:iam::111222333444:role/my-example-administrator-role`.

   Apply the following resource-based policy, modified for your environment, to your KMS key.

   JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
    {
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws-eusc:iam::111122223333:role/my-example-
administrator-role"
        },
        "Action": "kms:CreateGrant",
        "Resource": "arn:aws-eusc:kms:us-
west-2:111122223333:key/1example-2222-3333-4444-999example",
        "Condition": {
            "StringEquals": {
               "kms:EncryptionContext:userpool-id": "us-west-2_EXAMPLE"
            }
        }
```

```
    },
    {
        "Sid": "Allow Lambda to decrypt",
        "Effect": "Allow",
        "Principal": {
            "AWS": "arn:aws-eusc:iam::111122223333:role/my-lambda-function-
role"
        },
        "Action": "kms:Decrypt",
        "Resource": "*"
    }]
}
```

3. Create a Lambda function for the custom sender trigger. Amazon Cognito uses the AWS encryption SDK to encrypt the secrets, temporary passwords and codes that authorize your users' API requests.

   a. Assign a Lambda execution role that has, at minimum, `kms:Decrypt` permissions for your KMS key.

   b. Compose Lambda function code to send your messages. The input event to your function contains a secret. In your function, decrypt the secret with the AWS Encryption SDK and process any relevant metadata. Then send the code, your own custom message, and destination phone number to the custom API that delivers your message.

   c. Add the AWS Encryption SDK to your Lambda function. For more information, see AWS Encryption SDK programming languages. To update the Lambda package, complete the following steps.

      i. Export your Lambda function as a .zip file in the AWS Management Console.

      ii. Open your function and add the AWS Encryption SDK. For more information and download links, see AWS Encryption SDK programming languages in the *AWS Encryption SDK Developer Guide*.

      iii. Zip your function with your SDK dependencies, and upload the function to Lambda. For more information, see Deploying Lambda functions as .zip file archives in the *AWS Lambda Developer Guide*.

4. Grant Amazon Cognito service principal `cognito-idp.amazonaws.com` access to invoke the Lambda function.

   The following AWS CLI command grants Amazon Cognito permission to invoke your Lambda function:

```
aws lambda add-permission --function-name lambda_arn --statement-id
 "CognitoLambdaInvokeAccess" --action lambda:InvokeFunction --principal cognito-
idp.amazonaws.com
```

5. Generate an UpdateUserPool API request with a `LambdaConfig` parameter that adds custom sender Lambda triggers. You can't add triggers of this type in the Amazon Cognito console. Custom sender triggers require `LambdaConfig` parameters of `KMSKeyID` and `CustomSMSSender` or `CustomEmailSender` (or both).

Custom email sender Lambda trigger

When you assign a custom email sender trigger to your user pool, Amazon Cognito invokes a Lambda function instead of its default behavior when a user event requires that it send an email message. With a custom sender trigger, your AWS Lambda function can send email notifications to your users through a method and provider that you choose. The custom code of your function must process and deliver all email messages from your user pool.

This trigger serves scenarios where you might want to have greater control over how your user pool sends email messages. Your Lambda function can customize the call to Amazon SES API operations, for example when you want to manage multiple verified identities or cross AWS Regions. Your function also might redirect messages to another delivery medium or third-party service.

To learn how to configure a custom email sender trigger, see Activating custom sender Lambda triggers.

Custom email sender Lambda trigger sources

The following table shows the triggering events for custom email trigger sources in your Lambda code.

TriggerSource value	Event
`CustomEmailSender_SignUp`	A user signs up and Amazon Cognito sends a welcome message.
`CustomEmailSender_Authentication`	A user signs in and Amazon Cognito sends an email OTP or MFA code.

TriggerSource value	Event
CustomEmailSender_ForgotPassword	A user requests a code to reset their password.
CustomEmailSender_ResendCode	A user requests a replacement account-confirmation code.
CustomEmailSender_UpdateUserAttribute	A user updates an email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomEmailSender_VerifyUserAttribute	A user creates a new email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomEmailSender_AdminCreateUser	You create a new user in your user pool and Amazon Cognito sends them a temporary password.
CustomEmailSender_AccountTakeOverNotification	Amazon Cognito detects an attempt to take over a user account and sends the user a notification.

Custom email sender Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
        "type": "customEmailSenderRequestV1",
        "code": "string",
        "clientMetadata": {
            "string": "string",
             . . .
            },
        "userAttributes": {
            "string": "string",
```

```
            . . .
        }
}
```

Custom email sender request parameters

type

The request version. For a custom email sender event, the value of this string is always `customEmailSenderRequestV1`.

code

The encrypted code that your function can decrypt and send to your user.

clientMetadata

One or more key-value pairs that you can provide as custom input to the custom email sender Lambda function trigger. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions. Amazon Cognito doesn't include data from the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the post authentication function.

> **Note**
>
> Amazon Cognito sends `ClientMetadata` to custom email trigger functions in events with the following trigger sources:
>
> - `CustomEmailSender_ForgotPassword`
> - `CustomEmailSender_SignUp`
> - `CustomEmailSender_Authentication`
>
> Amazon Cognito doesn't send `ClientMetadata` in trigger events with source `CustomEmailSender_AccountTakeOverNotification`.

userAttributes

One or more key-value pairs that represent user attributes.

Custom email sender response parameters

Amazon Cognito doesn't expect any additional return information in the custom email sender response. Your Lambda function must interpret the event and decrypt the code, then deliver the message contents. A typical function assembles an email message and directs it to a third-party SMTP relay.

Code example

The following Node.js example processes an email message event in your custom email sender Lambda function. This example assumes your function has two environment variables defined.

`KEY_ID`

The ID of the KMS key that you want to use to encrypt and decrypt your users' codes.

`KEY_ARN`

The Amazon Resource Name (ARN) of the KMS key that you want to use to encrypt and decrypt your users' codes.

To deploy this function

1. Install the latest version of NodeJS in your developer workspace.
2. Create a new NodeJS project in your workspace.
3. Initialize your project with `npm init -y`.
4. Create the script for the Lambda function: `touch index.mjs`.
5. Paste the contents of the below example into `index.mjs`.
6. Download the project dependency, AWS Encryption SDK: `npm install @aws-crypto/client-node`.
7. Zip the project directory into a file: `zip -r my_deployment_package.zip .`.
8. Deploy the ZIP file to your function.

This example function decrypts the code and, for sign-up events, simulates sending an email message to the user's email address.

```
import { KmsKeyringNode, buildClient, CommitmentPolicy } from '@aws-crypto/client-
node';
```

```
// Configure the encryption SDK client with the KMS key from the environment variables
const { encrypt, decrypt } = buildClient(
    CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
);

const generatorKeyId = process.env.KEY_ID;
const keyIds = [process.env.KEY_ARN];
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds });

// Example function to simulate sending email.
// This example logs message details to CloudWatch Logs from your Lambda function.
// Update this function with custom logic that sends an email message to 'emailaddress'
 with body 'message'.
const sendEmail = async (emailAddress, message) => {
    // Log the destination with the email address masked.
    console.log(`Simulating email send to ${emailAddress.replace(/[^@.]/g, '*')}`);
    // Log the message with the code masked.
    console.log(`Message content: ${message.replace(/\b\d{6,8}\b/g, '********')}`);
    // Simulate API delay
    await new Promise(resolve => setTimeout(resolve, 100));
    console.log('Email sent successfully');
    return true;
};

export const handler = async (event) => {
    try {
        // Decrypt the secret code using encryption SDK
        let plainTextCode;
        if (event.request.code) {
            const { plaintext, messageHeader } = await decrypt(keyring,
 Buffer.from(event.request.code, 'base64'));
            plainTextCode = Buffer.from(plaintext).toString('utf-8');
        }

        // Handle different trigger sources
        if (event.triggerSource == 'CustomEmailSender_SignUp') {
            const emailAddress = event.request.userAttributes.email;
            const message = `Welcome! Your verification code is: ${plainTextCode}`;
            await sendEmail(emailAddress, message);
        }
        else if (event.triggerSource == 'CustomEmailSender_ResendCode') {
            // Handle resend code
        }
        else if (event.triggerSource == 'CustomEmailSender_ForgotPassword') {
```

```
            // Handle forgot password
        }
        else if (event.triggerSource == 'CustomEmailSender_UpdateUserAttribute') {
            // Handle update attribute
        }
        else if (event.triggerSource == 'CustomEmailSender_VerifyUserAttribute') {
            // Handle verify attribute
        }
        else if (event.triggerSource == 'CustomEmailSender_AdminCreateUser') {
            // Handle admin create user
        }
        else if (event.triggerSource == 'CustomEmailSender_Authentication') {
            // Handle authentication
        }
        else if (event.triggerSource ==
 'CustomEmailSender_AccountTakeOverNotification') {
            // Handle account takeover notification
        }

        return;
    } catch (error) {
        console.error('Error in custom email sender:', error);
        throw error;
    }
};
```

Custom SMS sender Lambda trigger

When you assign a custom SMS sender trigger to your user pool, Amazon Cognito invokes a Lambda function instead of its default behavior when a user event requires that it send an SMS message. With a custom sender trigger, your AWS Lambda function can send SMS notifications to your users through a method and provider that you choose. The custom code of your function must process and deliver all SMS messages from your user pool.

This trigger serves scenarios where you might want to have greater control over how your user pool sends SMS messages. Your Lambda function can customize the call to Amazon SNS API operations, for example when you want to manage multiple origination IDs or cross AWS Regions. Your function also might redirect messages to another delivery medium or third-party service.

To learn how to configure a custom email sender trigger, see Activating custom sender Lambda triggers.

Custom SMS sender Lambda trigger sources

The following table shows the triggering event for custom SMS trigger sources in your Lambda code.

TriggerSource value	Event
CustomSMSSender_SignUp	A user signs up and Amazon Cognito sends a welcome message.
CustomSMSSender_ForgotPassword	A user requests a code to reset their password.
CustomSMSSender_ResendCode	A user requests a new code to confirm their registration.
CustomSMSSender_VerifyUserAttribute	A user creates a new email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomSMSSender_UpdateUserAttribute	A user updates an email address or phone number attribute and Amazon Cognito sends a code to verify the attribute.
CustomSMSSender_Authentication	A user signs in and Amazon Cognito sends an SMS OTP or MFA code.
CustomSMSSender_AdminCreateUser	You create a new user in your user pool and Amazon Cognito sends them a temporary password.

Custom SMS sender Lambda trigger parameters

The request that Amazon Cognito passes to this Lambda function is a combination of the parameters below and the common parameters that Amazon Cognito adds to all requests.

JSON

```
{
    "request": {
```

```
        "type": "customSMSSenderRequestV1",
        "code": "string",
        "clientMetadata": {
            "string": "string",
             . . .
            },
        "userAttributes": {
            "string": "string",
            . . .
         }
}
```

Custom SMS sender request parameters

type

The request version. For a custom SMS sender event, the value of this string is always `customSMSSenderRequestV1`.

code

The encrypted code that your function can decrypt and send to your user.

clientMetadata

One or more key-value pairs that you can provide as custom input to the custom SMS sender Lambda function trigger. To pass this data to your Lambda function, you can use the ClientMetadata parameter in the AdminRespondToAuthChallenge and RespondToAuthChallenge API actions. Amazon Cognito doesn't include data from the ClientMetadata parameter in AdminInitiateAuth and InitiateAuth API operations in the request that it passes to the post authentication function.

userAttributes

One or more key-value pairs that represent user attributes.

Custom SMS sender response parameters

Amazon Cognito doesn't expect any additional return information in the response. Your function can use API operations to query and modify your resources, or record event metadata to an external system.

Code example

The following Node.js example processes an SMS message event in your custom SMS sender Lambda function. This example assumes your function has two environment variables defined.

`KEY_ID`

The ID of the KMS key that you want to use to encrypt and decrypt your users' codes.

`KEY_ARN`

The Amazon Resource Name (ARN) of the KMS key that you want to use to encrypt and decrypt your users' codes.

To deploy this function

1. Install the latest version of NodeJS in your developer workspace.
2. Create a new NodeJS project in your workspace.
3. Initialize your project with `npm init -y`.
4. Create the script for the Lambda function: `touch index.mjs`.
5. Paste the contents of the below example into `index.mjs`.
6. Download the project dependency, AWS Encryption SDK: `npm install @aws-crypto/client-node`.
7. Zip the project directory into a file: `zip -r my_deployment_package.zip .`.
8. Deploy the ZIP file to your function.

```
import { KmsKeyringNode, buildClient, CommitmentPolicy } from '@aws-crypto/client-
node';

// Configure the encryption SDK client with the KMS key from the environment variables
const { encrypt, decrypt } = buildClient(
    CommitmentPolicy.REQUIRE_ENCRYPT_ALLOW_DECRYPT
);

const generatorKeyId = process.env.KEY_ID;
const keyIds = [process.env.KEY_ARN];
const keyring = new KmsKeyringNode({ generatorKeyId, keyIds });

// Example function to simulate sending SMS.
```

```
// This example logs message details to CloudWatch Logs from your Lambda function.
// Update this function with custom logic that sends an SMS message to 'phoneNumber'
 with body 'message'.
const sendSMS = async (phoneNumber, message) => {
    // Log the destination with the phone number masked.
    console.log(`Simulating SMS send to ${phoneNumber.replace(/[^+]/g, '*')}`);
    // Log the message with the code masked.
    console.log(`Message content: ${message.replace(/\b\d{6,8}\b/g, '********')}`);
    // Simulate API delay
    await new Promise(resolve => setTimeout(resolve, 100));
    console.log('SMS sent successfully');
    return true;
};

export const handler = async (event) => {
    try {
        // Decrypt the secret code using encryption SDK
        let plainTextCode;
        if (event.request.code) {
            const { plaintext, messageHeader } = await decrypt(keyring,
 Buffer.from(event.request.code, 'base64'));
            plainTextCode = Buffer.from(plaintext).toString('utf-8');
        }

        // Handle different trigger sources
        if (event.triggerSource == 'CustomSMSSender_SignUp') {
            const phoneNumber = event.request.userAttributes.phone_number;
            const message = `Welcome! Your verification code is: ${plainTextCode}`;
            await sendSMS(phoneNumber, message);
        }
        else if (event.triggerSource == 'CustomSMSSender_ResendCode') {
            // Handle resend code
        }
        else if (event.triggerSource == 'CustomSMSSender_ForgotPassword') {
            // Handle forgot password
        }
        else if (event.triggerSource == 'CustomSMSSender_UpdateUserAttribute') {
            // Handle update attribute
        }
        else if (event.triggerSource == 'CustomSMSSender_VerifyUserAttribute') {
            // Handle verify attribute
        }
        else if (event.triggerSource == 'CustomSMSSender_AdminCreateUser') {
            // Handle admin create user
```

```
        }
        return;
    } catch (error) {
        console.error('Error in custom SMS sender:', error);
        throw error;
    }
};
```

Topics

- Evaluate SMS message capabilities with a custom SMS sender function

Evaluate SMS message capabilities with a custom SMS sender function

A custom SMS sender Lambda function accepts the SMS messages that your user pool would send, and the function delivers the content based on your custom logic. Amazon Cognito sends the Custom SMS sender Lambda trigger parameters to your function. Your function can do what you want with this information. For example, you can send the code to an Amazon Simple Notification Service (Amazon SNS) topic. An Amazon SNS topic subscriber can be an SMS message, an HTTPS endpoint, or an email address.

To create a test environment for Amazon Cognito SMS messaging with a custom SMS sender Lambda function, see amazon-cognito-user-pool-development-and-testing-with-sms-redirected-to-email in the aws-samples library on GitHub. The repository contains AWS CloudFormation templates that can create a new user pool, or work with a user pool that you already have. These templates create Lambda functions and an Amazon SNS topic. The Lambda function that the template assigns as a custom SMS sender trigger, redirects your SMS messages to the subscribers to the Amazon SNS topic.

When you deploy this solution to a user pool, all messages that Amazon Cognito usually sends through SMS messaging, the Lambda function instead sends to a central email address. Use this solution to customize and preview SMS messages, and to test the user pool events that cause Amazon Cognito to send an SMS message. After you complete your tests, roll back the CloudFormation stack, or remove the custom SMS sender function assignment from your user pool.

> **Important**
>
> Don't use the templates in amazon-cognito-user-pool-development-and-testing-with-sms-redirected-to-email to build a production environment. The custom SMS sender Lambda function in the solution *simulates* SMS messages, but the Lambda function sends them

all to a single central email address. Before you can send SMS messages in a production Amazon Cognito user pool, you must complete the requirements shown at SMS message settings for Amazon Cognito user pools.

Messaging and analytics

Configure email and SMS messaging for your user pool, and set up Amazon Pinpoint analytics to track user engagement.

Topics

- Email settings for Amazon Cognito user pools
- SMS message settings for Amazon Cognito user pools
- Using Amazon Pinpoint for user pool analytics

Email settings for Amazon Cognito user pools

Certain events in your application can cause Amazon Cognito to email your users. For example, if you configure your user pool to require email verification, Amazon Cognito sends an email when a user signs up for a new account in your app or resets their password. Depending on the action that initiates the email, the email contains a verification code or a temporary password.

To handle email delivery, you can use either of the following options:

- The default email configuration that is built into the Amazon Cognito service.
- Your Amazon Simple Email Service (Amazon SES) configuration.

You can change your delivery option after you create your user pool.

Amazon Cognito sends email messages to your users with either a code that they can enter or a URL link that they can select. The following table shows the events that can generate an email message.

Message options

Activity	API operation	Delivery options	Format options	Customizable	Message template
Forgot password	ForgotPassword, AdminResetUserPassword	Email, SMS	code	Yes	**Verification message**
Invitation	AdminCreateUser	Email, SMS	code	Yes	**Invitation message**
Self-registration	SignUp, ResendConfirmationCode	Email, SMS	code, link	Yes	**Verification message**
Email address or phone number verification	UpdateUserAttributes, AdminUpdateUserAttributes, GetUserAttributeVerificationCode	Email, SMS	code	Yes	**Verification message**
Multi-factor authentication (MFA)	AdminInitiateAuth, InitiateAuth	Email[1], SMS, authenticator app	code	Yes[2]	**MFA message**
One-time password authentication (OTP)	AdminInitiateAuth, InitiateAuth	Email[1], SMS	code	Yes	**MFA message**[3]

[1] Requires Essentials feature plan or higher and Amazon SES email configuration.

[2] For SMS and email messages.

[3] You can only customize the MFA message template when MFA is required or optional in your user pool. When MFA is inactive, Amazon Cognito sends one-time passwords with the default template.

Amazon SES charges for email messages. For more information, see Amazon SES pricing.

To learn more about email MFA, see SMS and email message MFA.

Amazon Cognito might prevent delivery of additional email or SMS messages to a single destination in a short time period. If you believe your user pool is affected, configure and review logs for message delivery errors and then contact your account team.

Default email configuration

Amazon Cognito can use its default email configuration to handle email deliveries for you. When you use the default option, Amazon Cognito limits the number of emails it sends each day for your user pool. For information on service limits, see Quotas in Amazon Cognito. For typical production environments, the default email limit is below the required delivery volume. To enable a higher delivery volume, you can use your Amazon SES email configuration.

When you use the default configuration, you use Amazon SES resources that are managed by AWS to send email messages. Amazon SES adds email addresses that return a hard bounce to an account-level suppression list or a global suppression list. If an undeliverable email address becomes deliverable later, you can't control its removal from the suppression list while your user pool is configured to use the default configuration. An email address can remain on the AWS-managed suppression list indefinitely. To manage undeliverable email addresses, use your Amazon SES email configuration with an account-level suppression list, as described in the next section.

When you use the default email configuration, you can use either of the following email addresses as the FROM address:

- The default email address, *no-reply@verificationemail.com*.
- A custom email address. Before you can use your own email address, you must verify it with Amazon SES and grant Amazon Cognito permission to use this address.

Amazon SES email configuration

Your application might require a higher delivery volume than what is available with the default option. To increase the possible delivery volume, use your Amazon SES resources with your user pool to email your users. You can also monitor your email sending activity when you send email messages with your own Amazon SES configuration.

Before you can use your Amazon SES configuration, you must verify one or more email addresses, or a domain, with Amazon SES. Use a verified email address, or an address from a verified domain,

as the FROM email address that you assign to your user pool. When Amazon Cognito sends email to a user, it calls Amazon SES for you and uses your email address.

When you use your Amazon SES configuration, the following conditions apply:

- The email delivery limits for your user pool are the same limits that apply to your Amazon SES verified email address in your AWS account.
- You can manage your messages to undeliverable email addresses with an account-level suppression list in Amazon SES that overrides the global suppression list. When you use an account-level suppression list, email message bounces affect the reputation of your account as a sender. For more information, see Using the Amazon SES account-level suppression list in the Amazon Simple Email Service Developer Guide.

Amazon SES email configuration Regions

The AWS Region where you create a user pool will have one of three requirements for the configuration of email messages with Amazon SES. You might send email messages from Amazon SES in the same Region as your user pool, several Regions including the same Region, or one or more remote Regions. For best performance, send email messages with a Amazon SES verified identity in the same Region as your user pool when you have the option.

Categories of Region requirements for Amazon SES verified identities

In-Region only

Your user pools can send email messages with verified identities in the same AWS Region as the user pool. In the default email configuration without a custom `FROM` email address, Amazon Cognito uses a `no-reply@verificationemail.com` verified identity in the same Region.

Backwards compatible

Your user pools can send email messages with verified identities in the same AWS Region or in one of the following alternate Regions:

- US East (N. Virginia)
- US West (Oregon)
- Europe (Ireland)

This feature supports continuity for user pool resources that you might have created to match Amazon Cognito requirements when the service launched. User pools from that period could

only send email messages with verified identities in a limited number of AWS Regions. In the default email configuration without a custom FROM email address, Amazon Cognito uses a no-reply@verificationemail.com verified identity in the same Region.

Alternate Region

Your user pools can send email messages with verified identities in an alternate AWS Region that is outside of the user pool Region. This configuration occurs when Amazon SES isn't available in a Region where Amazon Cognito is available.

The Amazon SES sending authorization policy for your verified identity in the alternate Region must trust the Amazon Cognito service principal of the originating Region. For more information, see To grant permissions to use the default email configuration.

In some of these Regions, Amazon Cognito splits email messages between two alternate Regions for the default email configuration of COGNITO_DEFAULT. In these cases, to use a custom FROM email address, the Amazon SES sending authorization policy for your verified identity in each alternate Region must trust the Amazon Cognito service principal of the originating Region. For more information, see To grant permissions to use the default email configuration. With the Amazon SES email configuration of DEVELOPER in these Regions, you must use a verified identity in the *first* listed Region and configure it to trust the Amazon Cognito service principal in the user pool Region. For example, in a user pool in Middle East (UAE), configure a verified identity in Europe (Frankfurt) to trust cognito-idp.me-central-1.amazonaws.com. In the default email configuration without a custom FROM email address, Amazon Cognito uses a no-reply@verificationemail.com verified identity in each Region.

Note

Under the following combination of conditions, you must specify the SourceArn parameter of EmailConfiguration with a wildcard in the Region element, in the format arn:*${Partition}*:ses:*:*${Account}*:identity/*${IdentityName}*. This permits your user pool to send email messages with identical verified identities in your AWS account in both AWS Regions.

- Your EmailSendingAccount is COGNITO_DEFAULT.
- You want to use a custom FROM address.
- Your user pool sends emails in an **Alternate Region**.

- Your user pool has a *second*[1] **Alternate Region** specified in the table of **Amazon SES supported Regions** that follows.

If you create a user pool programmatically–with an AWS SDK, the Amazon Cognito API or CLI, the AWS CDK, or AWS CloudFormation–your user pool sends email messages with the Amazon SES identity that the `SourceArn` parameter of EmailConfiguration specifies for your user pool. The Amazon SES identity must occupy a supported AWS Region. If your `EmailSendingAccount` is `COGNITO_DEFAULT` and you don't specify a `SourceArn` parameter, Amazon Cognito sends email messages from `no-reply@verificationemail.com` using resources in the Region where you created your user pool.

The following table shows the AWS Regions where you can use Amazon SES identities with Amazon Cognito.

User pool Region	Region option	Amazon SES supported Regions
US East (N. Virginia)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)
US East (Ohio)	Backwards compatible	US East (Ohio), US East (N. Virginia), US West (Oregon), Europe (Ireland)
US West (N. California)	In-Region only	US West (N. California)
US West (Oregon)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)
Canada (Central)	Backwards compatible	Canada (Central), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Canada West (Calgary)	Alternate Region	Canada (Central), US West (N. California)[1]

User pool Region	Region option	Amazon SES supported Regions
Mexico (Central)	Alternate Region	US East (N. Virginia), US West (Oregon)[1]
Asia Pacific (Tokyo)	Backwards compatible	Asia Pacific (Tokyo), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Hong Kong)	Alternate Region	Asia Pacific (Singapore), Asia Pacific (Tokyo)[1]
Asia Pacific (Seoul)	Backwards compatible	Asia Pacific (Seoul), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Malaysia)	Alternate Region	Asia Pacific (Sydney), Asia Pacific (Singapore)[1]
Asia Pacific (Thailand)	Alternate Region	Asia Pacific (Singapore), Asia Pacific (Mumbai)[1]
Asia Pacific (Mumbai)	Backwards compatible	Asia Pacific (Mumbai), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Hyderabad)	Alternate Region	Asia Pacific (Mumbai), Asia Pacific (Singapore)[1]
Asia Pacific (Singapore)	Backwards compatible	Asia Pacific (Singapore), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Asia Pacific (Sydney)	Backwards compatible	Asia Pacific (Sydney), US East (N. Virginia), US West (Oregon), Europe (Ireland)

User pool Region	Region option	Amazon SES supported Regions
Asia Pacific (Osaka)	In-Region only	Asia Pacific (Osaka)
Asia Pacific (Jakarta)	In-Region only	Asia Pacific (Jakarta)
Asia Pacific (Melbourne)	Alternate Region	Asia Pacific (Sydney), Asia Pacific (Singapore)[1]
Europe (Ireland)	Backwards compatible	US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (London)	Backwards compatible	Europe (London), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (Paris)	In-Region only	Europe (Paris)
Europe (Frankfurt)	Backwards compatible	Europe (Frankfurt), US East (N. Virginia), US West (Oregon), Europe (Ireland)
Europe (Zurich)	Alternate Region	Europe (Frankfurt), Europe (London)[1]
Europe (Stockholm)	In-Region only	Europe (Stockholm)
Europe (Milan)	In-Region only	Europe (Milan)
Europe (Spain)	Alternate Region	Europe (Paris), Europe (Stockholm)[1]
Middle East (Bahrain)	In-Region only	Middle East (Bahrain)
Middle East (UAE)	Alternate Region	Europe (Frankfurt), Europe (London)[1]
South America (São Paulo)	In-Region only	South America (São Paulo)

User pool Region	Region option	Amazon SES supported Regions
Israel (Tel Aviv)	In-Region only	Israel (Tel Aviv)
Africa (Cape Town)	In-Region only	Africa (Cape Town)

[1] Used in user pools with the default email configuration. Amazon Cognito distributes email messages among verified identities with the same email address in each Region. To use a custom FROM address, configure `EmailConfiguration` with a `SourceArn` parameter in the format `arn:${Partition}:ses:*:${Account}:identity/${IdentityName}`.

Configuring email for your user pool

Complete the following steps to configure the email settings for your user pool. Depending on the settings that you use, you might need IAM permissions in Amazon SES, AWS Identity and Access Management (IAM), and Amazon Cognito.

> **Note**
>
> You can't share the resources that you create in these steps across AWS accounts. For example, you can't configure a user pool in one account, and then use it with an Amazon SES email address in a different account. If you use Amazon Cognito in multiple accounts, repeat these steps for each account.

Step 1: Verify your email address or domain with Amazon SES

Before you configure your user pool, you must verify one or more domains or email addresses with Amazon SES if you want to do either of the following:

- Use your own email address as the FROM address
- Use your Amazon SES configuration to handle email delivery

By verifying your email address or domain, you confirm that you own it, which helps prevent unauthorized use.

For information on verifying an email address with Amazon SES, see Verifying an Email Address in the *Amazon Simple Email Service Developer Guide*. For information on verifying a domain with Amazon SES, see Verifying domains.

Step 2: Move your account out of the Amazon SES sandbox

Omit this step if you are using the default Amazon Cognito email configuration.

When you first use Amazon SES in any AWS Region, it places your AWS account in the Amazon SES sandbox for that Region. Amazon SES uses the sandbox to prevent fraud and abuse. If you use your Amazon SES configuration to handle email delivery, you must move your AWS account out of the sandbox before Amazon Cognito can email your users.

In the sandbox, Amazon SES imposes restrictions on how many emails you can send and where you can send them. You can send emails only to addresses and domains that you have verified with Amazon SES, or you can send them to Amazon SES mailbox simulator addresses. While your AWS account remains in the sandbox, don't use your Amazon SES configuration for applications that are in production. In this situation, Amazon Cognito can't send messages to your users' email addresses.

To remove your AWS account from the sandbox, see Moving out of the Amazon SES sandbox in the *Amazon Simple Email Service Developer Guide.*

Step 3: Grant email permissions to Amazon Cognito

You might need to grant specific permissions to Amazon Cognito before it can email your users. The permissions that you grant, and the process that you use to grant them, depend on whether you are using the default email configuration, or your Amazon SES configuration.

To grant permissions to use the default email configuration

Complete this step only if you configure your user pool to **Send email with Cognito** or set `EmailSendingAccount` to `COGNITO_DEFAULT`.

With the default email configuration, your user pool can send email messages with either of the following addresses.

- The default address `no-reply@verificationemail.com`.
- A custom FROM address from your verified email addresses or domains in Amazon SES.

If you use a custom address, Amazon Cognito needs additional permissions to email users from that address. These permissions are granted by a sending authorization policy for the address or domain in Amazon SES. If you use the Amazon Cognito console to add a custom address to your user pool, the policy is automatically attached to the Amazon SES verified email address. However, if you configure your user pool outside of the console, such as using the AWS CLI or the Amazon Cognito API, you must attach the policy using the Amazon SES console or the PutIdentityPolicy API.

> **Note**
>
> You can only configure a FROM address in a verified domain using the AWS CLI or the Amazon Cognito API.

A sending authorization policy allows or denies access based on the account resources that are using Amazon Cognito to invoke Amazon SES. For more information about resource-based policies, see the IAM User Guide. You can also find example resource-based policies in the Amazon SES Developer Guide.

Example Sending authorization policy

The following example sending authorization policy grants Amazon Cognito a limited ability to use an Amazon SES verified identity. Amazon Cognito can only send email messages when it does so on behalf of both the user pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

Regions with Amazon SES

Your sending authorization policy in the user pool Region or alternate Region must permit the Amazon Cognito service principal to send email messages. Refer to the Regions table for more information. If your **User pool Region** matches at least one value in **Amazon SES Region**, configure your sending authorization policy with the global service principal in the following example.

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "stmnt1234567891234",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "email.cognito-idp.amazonaws.com"
                ]
            },
            "Action": [
                "SES:SendEmail",
                "SES:SendRawEmail"
            ],
            "Resource": "arn:aws:ses:eusc-de-
east-1:111122223333:identity/support@example.com",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/us-east-1_EXAMPLE"
                }
            }
        }
    ]
}
```

Opt-in Regions without Amazon SES

Amazon SES isn't available in all opt-in AWS Regions where Amazon Cognito is available. Middle East (UAE) is an example, and can only send emails with verified identities in Europe (Frankfurt) (`eu-central-1`). In user pools with the default email configuration, Amazon Cognito also sends email messages with a verified identity in each of two Regions. In the case of Middle East (UAE), the additional Region is Europe (London). You must update the sending authorization policy in both Regions.

Your sending authorization policy in each of the alternate Regions must permit the Amazon Cognito service principal in the user pool opt-in Region to send email messages. Refer to the Regions table for more information. If your Region is marked as **Alternate Region**, configure your sending authorization policies with the Regional service principal as in the following

example. Replace the example Region identifier *me-central-1* with the required Region ID as needed.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.me-central-1.amazonaws.com"
                ]
            },
            "Action": [
                "SES:SendEmail",
                "SES:SendRawEmail"
            ],
            "Resource": "arn:aws:ses:eusc-de-
east-1:111122223333:identity/support@example.com",
            "Condition": {
                "StringEquals": {
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/us-east-1_EXAMPLE"
                }
            }
        }
    ]
}
```

For more information about policy syntax, see Amazon SES sending authorization policies in the *Amazon Simple Email Service Developer Guide*.

For more examples, see Amazon SES sending authorization policy examples in the *Amazon Simple Email Service Developer Guide*.

To grant permissions to use your Amazon SES configuration

If you configure your user pool to use your Amazon SES configuration, Amazon Cognito needs additional permissions to call Amazon SES on your behalf when it emails your users. This authorization is granted with the IAM service.

When you configure your user pool with this option, Amazon Cognito creates a *service-linked role*, which is a type of IAM role, in your AWS account. This role contains the permissions that allow Amazon Cognito to access Amazon SES and send email with your address.

Amazon Cognito creates your service-linked role with the AWS credentials of the user session that sets the configuration. The IAM permissions of this session must include the `iam:CreateServiceLinkedRole` action. For more information about permissions in IAM, see Access management for AWS resources in the *IAM User Guide*.

For more information about the service-linked role that Amazon Cognito creates, see Using service-linked roles for Amazon Cognito.

Step 4: Configure your user pool

Complete the following steps if you want to configure your user pool with any of the following:

- A custom FROM address that appears as the email sender
- A custom REPLY-TO address that receives the messages that your users send to your FROM address
- Your Amazon SES configuration

> **Note**
>
> If your verified identity is an email address, Amazon Cognito sets that email address as the FROM and REPLY-TO email address by default. But, if your verified identity is a domain, you must provide a value for the FROM email address.

Omit this procedure if you want to use the default Amazon Cognito email configuration and address.

To configure your user pool to use a custom email address

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Authentication methods** menu, locate **Email configuration**, choose **Edit**.
5. On the **Edit email configuration** page, select **Send email from Amazon SES** or **Send email with Amazon Cognito**. You can customize the **SES Region**, **Configuration Set**, and **FROM sender name** only when you choose **Send email from Amazon SES**.
6. To use a custom FROM address, complete the following steps:

   a. Under **SES Region**, choose the Region that contains your verified email address.

   b. Under **FROM email address**, choose your email address. Use an email address that you have verified with Amazon SES.

   c. (Optional) Under **Configuration set**, choose a configuration set for Amazon SES to use. Making and saving this change creates a service-linked role.

   d. (Optional) Under **FROM sender address**, enter an email address. You can provide only an email address, or an email address and a friendly name in the format `Jane Doe <janedoe@example.com>`.

   e. (Optional) Under **REPLY-TO email address**, enter the email address where you want to receive messages that your users send to your FROM address.
7. Choose **Save changes**.

Related Topics

- Customizing email verification messages
- Customizing user invitation messages

SMS message settings for Amazon Cognito user pools

Some Amazon Cognito events for your user pool might cause Amazon Cognito to send SMS text messages to your users. For example, if you configure your user pool to require phone verification, Amazon Cognito sends an SMS text message when a user signs up for a new account in your app or resets their password. Depending on the action that initiates the SMS text message, the message contains a verification code, a temporary password, or a welcome message.

Amazon Cognito can send SMS messages with one of the following configurations:

- **Amazon Simple Notification Service (Amazon SNS)** – Amazon Cognito calls Amazon SNS, which routes SMS messages to AWS End User Messaging SMS.
- **AWS End User Messaging SMS direct** – Amazon Cognito calls AWS End User Messaging SMS directly, without Amazon SNS as an intermediary. You can use this option in most AWS Regions where Amazon Cognito is available. For exceptions, see AWS End User Messaging SMS direct path Region configuration.

A user pool uses one SMS configuration at a time. The Amazon SNS and AWS End User Messaging SMS direct configurations are mutually exclusive. When you set one configuration, Amazon Cognito clears the other. This applies to CreateUserPool, UpdateUserPool, and SetUserPoolMfaConfig operations.

Amazon SNS SMS is available in the AWS Regions listed in Amazon SNS path Region configuration. To send SMS messages from a user pool in any other AWS Region, use AWS End User Messaging SMS directly.

If you send a text message through Amazon Cognito for the first time, AWS End User Messaging SMS places you in a sandbox environment. In the sandbox environment, you can test your applications for SMS text messages. In the sandbox, you can send messages to a limited number of verified destination phone numbers if you have an origination identity, or simulate sending messages if you don't.

> **Note**
>
> In November 2024, AWS replaced Amazon SNS SMS messaging with AWS End User Messaging SMS. When you use the Amazon SNS configuration path, user pools initiate SMS messages with the Amazon SNS Publish operation, which is a pass-through to AWS End User Messaging SMS. For this path, you must configure permissions for `sns:Publish`. When you use the direct AWS End User Messaging SMS configuration path, Amazon Cognito calls the AWS End User Messaging SMS `SendTextMessage` operation and you must configure permissions for `sms-voice:SendTextMessage`.

AWS End User Messaging SMS charges for SMS text messages. For more information, see AWS End User Messaging SMS pricing.

Amazon Cognito sends SMS messages to your users with a code that they can enter. The following table shows the events that can generate an SMS message.

Message options

Activity	API operation	Delivery options	Format options	Customizable	Message template
Forgot password	ForgotPassword, AdminResetUserPassword	Email, SMS	code	Yes	**Verification message**
Invitation	AdminCreateUser	Email, SMS	code	Yes	**Invitation message**
Self-registration	SignUp, ResendConfirmationCode	Email, SMS	code, link	Yes	**Verification message**
Email address or phone number verification	UpdateUserAttributes, AdminUpdateUserAttributes, GetUserAttributeVerificationCode	Email, SMS	code	Yes	**Verification message**
Multi-factor authentication (MFA)	AdminInitiateAuth, InitiateAuth	Email[1], SMS, authenticator app	code	Yes[2]	**MFA message**
One-time password authentication (OTP)	AdminInitiateAuth, InitiateAuth	Email[1], SMS	code	Yes	**MFA message**[3]

[1] Requires Essentials feature plan or higher and Amazon SES email configuration.

[2] For SMS and email messages.

[3] You can only customize the MFA message template when MFA is required or optional in your user pool. When MFA is inactive, Amazon Cognito sends one-time passwords with the default template.

AWS End User Messaging SMS charges for SMS messages. For more information, see AWS End User Messaging SMS pricing.

To learn more about MFA, see SMS and email message MFA.

Amazon Cognito might prevent delivery of additional email or SMS messages to a single destination in a short time period. If you believe your user pool is affected, configure and review logs for message delivery errors and then contact your account team.

Choosing between Amazon SNS and AWS End User Messaging SMS

Amazon Cognito user pools can send SMS messages through two paths: Amazon SNS or AWS End User Messaging SMS directly. For new user pools, use the AWS End User Messaging SMS direct path. Amazon SNS SMS is available only in a fixed set of AWS Regions. The direct AWS End User Messaging SMS path supports additional Regions and exposes features that the Amazon SNS path does not. Both paths deliver messages through AWS End User Messaging SMS.

The following table compares the two SMS configuration paths.

Capability	Amazon SNS path	AWS End User Messaging SMS direct path
Region availability	Available in the AWS Regions listed in the Amazon SNS path Region table	Available in most AWS Regions where Amazon Cognito is available. Regions where only the Amazon SNS path is available are listed in AWS End User Messaging SMS direct path Region configuration.
Integration	Amazon Cognito calls the Amazon SNS Publish operation, which passes through to AWS End User Messaging SMS	Amazon Cognito calls the AWS End User Messaging SMS SendTextMessage operation directly

Capability	Amazon SNS path	AWS End User Messaging SMS direct path
IAM permission	`sns:Publish`	`sms-voice:SendTextMessage`
Configuration sets	Not configurable through Amazon Cognito	Set the `ConfigurationSetName` field for event logging and destinations
Origination identity	Not configurable through Amazon Cognito	Set the `OriginationIdentity` field
Cross-Region sending	Set the `SnsRegion` field	Not supported
India DLT registration	Not configurable through Amazon Cognito	Set the `InEntityId` and `InTemplateId` fields
Fraud protection (Protect)	Account-default protect configuration applies	Account-default applies. You can also apply a protect configuration to a specific use case through a configuration set.

Note

A default AWS End User Messaging SMS Protect configuration applies to all messages, including those from Amazon Cognito. To apply a protect configuration to a specific use case, link it to a configuration set and use the direct AWS End User Messaging SMS path. For more information, see Fraud Protection in AWS End User Messaging SMS.

SMS configuration in the Amazon Cognito user pools API

The SmsConfigurationType object in the Amazon Cognito user pools API contains the SMS configuration for your user pool. This object has the following configuration paths:

Amazon SNS path

Set `SnsCallerArn` to the ARN of an IAM role that grants Amazon Cognito permission to call `sns:Publish`. Optionally set `SnsRegion` and `ExternalId`. The `SnsCallerArn` parameter is optional when you use the AWS End User Messaging SMS direct path.

AWS End User Messaging SMS direct path

Set the `EumsSms` object of type `EumsSmsConfigurationType`. This object has the following members:

- `CallerArn` (Required) – The ARN of an IAM role that grants Amazon Cognito permission to call `sms-voice:SendTextMessage`.
- `ExternalId` (Optional) – The external ID for the IAM role trust policy. This value provides confused-deputy prevention.
- `OriginationIdentity` (Optional) – The phone number, pool, or sender ID to use for sending.
- `ConfigurationSetName` (Optional) – The AWS End User Messaging SMS configuration set to use.
- `InEntityId` (Optional) – The entity registration ID from India DLT registration. Required for sending to recipients in India.
- `InTemplateId` (Optional) – The template registration ID from India DLT registration. Required for sending to recipients in India.
- `Region` (Optional) – The AWS Region of your AWS End User Messaging SMS resources. For more information, see Choose the AWS Region for SMS messages.

For more information, see SmsConfigurationType in the *Amazon Cognito User Pools API Reference*.

Best practices

Because of the volume of unsolicited SMS traffic worldwide, some governments impose barriers between the senders and recipients of SMS messages. When you use SMS messages for MFA and user updates, you must take additional steps to ensure that your messages are delivered. You must also monitor SMS-message-related regulations in countries where your users might live and keep your SMS message configuration current. For more information, see SMS and MMS country capabilities and limitations in the *AWS End User Messaging SMS User Guide*.

The use of SMS messages to authenticate and verify users isn't a security best practice. Phone numbers can change owners, and might not reliably represent a *something you have* factor of MFA

for your users. Instead, implement TOTP MFA in your app or with your third-party IdP. You can also create additional custom authentication factors with Custom authentication challenge Lambda triggers.

Review the following links for information about securing your SMS message delivery architecture.

- Reduce risks of user sign-up fraud and SMS pumping with Amazon Cognito user pools
- Defending Against SMS Pumping: New AWS Features to Help Combat Artificially Inflated Traffic

Setting up SMS messaging for the first time in Amazon Cognito user pools

Amazon Cognito uses either Amazon SNS (and indirectly AWS End User Messaging SMS) or AWS End User Messaging SMS directly to send SMS messages from your user pools. You can also use a Custom SMS sender Lambda trigger to use your own resources to send SMS messages. The first time that you set up SMS text messages in a particular AWS Region, AWS End User Messaging SMS places your AWS account in the SMS sandbox for that Region. AWS End User Messaging SMS uses the sandbox to prevent fraud and abuse and to meet compliance requirements. When your AWS account is in the sandbox, AWS End User Messaging SMS imposes some restrictions. For example, you can send text messages to a maximum of 10 verified destination numbers if you have an origination identity, or you can simulate sending messages without an origination identity. While your AWS account remains in the sandbox, do not send SMS messages in production. When you're in the sandbox, Amazon Cognito can't send messages to your users' phone numbers.

Topics

- Prepare an IAM role that Amazon Cognito can use to send SMS messages with AWS End User Messaging SMS
- To set up AWS End User Messaging SMS direct SMS for a user pool
- Choose the AWS Region for SMS messages
- Obtain an origination identity to send SMS messages to US phone numbers
- Confirm that you are in the SMS sandbox
- Move your account out of the sandbox
- Use simulator numbers or verified phone numbers with AWS End User Messaging SMS
- Complete user pool setup in Amazon Cognito

Prepare an IAM role that Amazon Cognito can use to send SMS messages with AWS End User Messaging SMS

When you send an SMS message from your user pool, Amazon Cognito assumes an IAM role in your account. The permissions that the role requires depend on the SMS configuration path you choose:

- **Amazon SNS path** – The role must grant the `sns:Publish` permission. Amazon Cognito uses this permission to send SMS messages through Amazon SNS.
- **AWS End User Messaging SMS direct path** – The role must grant the `sms-voice:SendTextMessage` permission. Amazon Cognito uses this permission to call AWS End User Messaging SMS directly.

In the Amazon Cognito console, you can set an **IAM role selection** from the **Authentication methods** menu of your user pool, under **SMS** or make this selection during the user pool creation wizard.

The following example IAM role trust policy grants Amazon Cognito user pools a limited ability to assume the role. Amazon Cognito can only assume the role when it meets the following conditions:

- The assume-role operation is on behalf of the user pool in the `aws:SourceArn` condition.
- The assume-role operation is on behalf of a user pool in the AWS account set by the `aws:SourceAccount` condition.
- The assume-role operation includes the external ID in the `sts:externalId` condition.

These trust policy conditions apply to both the Amazon SNS path and the AWS End User Messaging SMS direct path.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "",
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-idp.amazonaws.com"
            },
```

```
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "sts:ExternalId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
                    "aws:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "aws:SourceArn": "arn:aws-eusc:cognito-idp:us-
west-2:111122223333:userpool/us-west-2_EXAMPLE"
                }
            }
        }
    ]
}
```

You can specify an exact user pool ARN or a wildcard ARN in the value of the `aws:SourceArn` condition. Look up the ARNs of your user pools in the AWS Management Console or with a DescribeUserPool API request.

To send SMS messages for multi-factor authentication, your IAM role trust policy must have an `sts:ExternalId` condition. The value of this condition must match the `ExternalId` property of the SmsConfiguration of your user pool. When you create an IAM role during the process of user pool creation in the Amazon Cognito console, Amazon Cognito configures the external ID for you in the role and in the user pool settings. This isn't true when you use an existing IAM role.

You must update the user pool `ExternalId` parameter in an UpdateUserPool API request and update the IAM role trust policy with an `sts:externalId` condition with the same value. To learn how to use the API to update a user pool in a way that preserves the original configuration, see Updating user pool and app client configuration.

For more information about IAM roles and trust policies, see Roles terms and concepts in the *AWS Identity and Access Management User Guide*.

To set up AWS End User Messaging SMS direct SMS for a user pool

Use the following procedure to configure your user pool to send SMS messages directly through AWS End User Messaging SMS. This configuration uses the `EumsSms` object in SmsConfigurationType.

To set up AWS End User Messaging SMS direct SMS messaging

1. Create or identify an AWS End User Messaging SMS origination identity in the AWS Region that you want to send from. An origination identity can be a phone number, sender ID, or pool. You can also create an AWS End User Messaging SMS configuration set to apply sending rules. For instructions on requesting a phone number, see Request a phone number in the *AWS End User Messaging SMS User Guide*.
2. Create an IAM role that Amazon Cognito can assume to call AWS End User Messaging SMS. The trust policy for this role is the same as the Amazon SNS path: it must include `aws:SourceArn`, `aws:SourceAccount`, and `sts:ExternalId` conditions. For trust policy details, see Prepare an IAM role that Amazon Cognito can use to send SMS messages with AWS End User Messaging SMS.
3. Attach the following identity (permissions) policy to the role. This policy grants Amazon Cognito permission to call `sms-voice:SendTextMessage` on the origination identity that your user pool sends from. Scope the `Resource` element to the ARN of your phone number, phone pool, or sender ID as a least-privilege practice. Replace the example ARN with the ARN of your origination identity.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "sms-voice:SendTextMessage"
            ],
            "Resource": "arn:aws:sms-voice:us-east-1:123456789012:phone-number/
phone-a1b2c3d4e5f6EXAMPLE"
        }
    ]
}
```

The origination identity ARN uses one of the following formats, depending on the identity type:

- Phone number – `arn:`*`partition`*`:sms-voice:`*`region`*`:`*`account`*`:phone-number/`*`phoneNumberId`*
- Phone pool – `arn:`*`partition`*`:sms-voice:`*`region`*`:`*`account`*`:pool/`*`poolId`*

- Sender ID – `arn:`*`partition`*`:sms-voice:`*`region`*`:`*`account`*`:sender-id/`*`senderId`*`/`*`isoCountryCode`*

> **Important**
>
> When your policy scopes the `Resource` element to an origination identity ARN, set the `OriginationIdentity` field to that same ARN—for example, `arn:aws:sms-voice:us-east-1:123456789012:phone-number/phone-a1b2c3d4e5f6EXAMPLE`. A phone number in E.164 format (such as `+12065550100`) doesn't match the scoped policy, so the request isn't authorized and Amazon Cognito can't send SMS messages.

4. If you send SMS messages to recipients in India, register a DLT entity ID and template ID with a Distributed Ledger Technology (DLT) portal. Supply the entity ID as `InEntityId` and the template ID as `InTemplateId` in the `EumsSms` configuration. For more information about India DLT registration, see Special requirements for sending SMS messages to recipients in India in the *AWS End User Messaging SMS User Guide*.
5. Test your configuration by calling the AWS End User Messaging SMS SendTextMessage API operation directly with the origination identity and IAM role that you configured. Confirm that your test message is delivered before you assign the configuration to your user pool.
6. Set the `EumsSms` object on your user pool. You can set this configuration in a CreateUserPool, UpdateUserPool, or SetUserPoolMfaConfig request. The `CallerArn` parameter is required. The following example shows the minimum `EumsSms` configuration:

```
"EumsSms": {
    "CallerArn": "arn:aws:iam::123456789012:role/service-role/CognitoEumsSmsRole",
    "ExternalId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

   To send from a specific origination identity or use a configuration set, include the `OriginationIdentity` and `ConfigurationSetName` parameters.

IAM permissions for AWS End User Messaging SMS direct SMS

The IAM role in `CallerArn` requires a trust policy that permits the `cognito-idp.amazonaws.com` service principal to assume it. Use the same trust policy structure as the

Amazon SNS path (see Prepare an IAM role that Amazon Cognito can use to send SMS messages with AWS End User Messaging SMS). The role also requires a permissions policy that grants `sms-voice:SendTextMessage` scoped to your origination identity ARN (see the setup procedure in To set up AWS End User Messaging SMS direct SMS for a user pool).

Choose the AWS Region for SMS messages

> **Note**
>
> SMS messages in AWS are now managed in AWS End User Messaging SMS.

Amazon SNS path Region configuration

In some AWS Regions, you can choose the Region that contains the Amazon SNS resources that you want to use for Amazon Cognito SMS messages. In any AWS Region where Amazon Cognito is available, except for Asia Pacific (Seoul), you can use Amazon SNS resources in the AWS Region where you created your user pool. To make your SMS messaging faster and more reliable when you have a choice of Regions, use Amazon SNS resources in the same Region as your user pool.

Choose a Region for SMS resources in the **Configure message delivery** step of the new user pool wizard. You can also choose **Edit** under **SMS** in the **Authentication methods** menu of an existing user pool.

At launch, for some AWS Regions, Amazon Cognito sent SMS messages with Amazon SNS resources in an alternate Region. To set your preferred Region, use the `SnsRegion` parameter of the SmsConfigurationType object for your user pool. When you programmatically create an Amazon Cognito user pools resource in an **Amazon Cognito Region** from the following table and you do not provide an `SnsRegion` parameter, your user pool can send SMS messages with Amazon SNS resources in a legacy **Amazon SNS Region**.

Amazon Cognito user pools in the Asia Pacific (Seoul) AWS Region must use your Amazon SNS configuration in the Asia Pacific (Tokyo) Region.

Amazon SNS (through AWS End User Messaging SMS) sets the spending quota for all new accounts at $1.00 (USD) per month. You might have increased your spend limit in an AWS Region that you use with Amazon Cognito. Before you change the AWS Region for Amazon SNS SMS messages, open a quota increase case in the AWS Support Center to increase your limit in the new Region. For more information, see Moving from the AWS End User Messaging SMS MMS and Voice sandbox to production in the *AWS End User Messaging SMS User Guide*.

You can send SMS messages for any **Amazon Cognito Region** in the following table with AWS End User Messaging SMS resources in the corresponding **SMS message Region**.

Amazon Cognito Region	SMS message Region
US East (Ohio)	US East (Ohio), US East (N. Virginia)
US East (N. Virginia)	US East (N. Virginia)
US West (N. California)	US West (N. California)
US West (Oregon)	US West (Oregon)
Canada (Central)	Canada (Central), US East (N. Virginia)
Canada West (Calgary)	Canada West (Calgary)
Mexico (Central)	Mexico (Central)
Europe (Frankfurt)	Europe (Frankfurt), Europe (Ireland)
Europe (London)	Europe (London), Europe (Ireland)
Europe (Ireland)	Europe (Ireland)
Europe (Paris)	Europe (Paris)
Europe (Stockholm)	Europe (Stockholm)
Europe (Milan)	Europe (Milan)
Europe (Spain)	Europe (Spain)
Europe (Zurich)	Europe (Zurich)
Asia Pacific (Malaysia)	Asia Pacific (Singapore)
Asia Pacific (Thailand)	Asia Pacific (Mumbai)
Asia Pacific (Mumbai)	Asia Pacific (Mumbai), Asia Pacific (Singapore)
Asia Pacific (Hyderabad)	Asia Pacific (Hyderabad)

Amazon Cognito Region	SMS message Region
Asia Pacific (Hong Kong)	Asia Pacific (Singapore)
Asia Pacific (Seoul)	Asia Pacific (Tokyo)
Asia Pacific (Singapore)	Asia Pacific (Singapore)
Asia Pacific (Sydney)	Asia Pacific (Sydney)
Asia Pacific (Tokyo)	Asia Pacific (Tokyo)
Asia Pacific (Jakarta)	Asia Pacific (Jakarta)
Asia Pacific (Osaka)	Asia Pacific (Osaka)
Asia Pacific (Melbourne)	Asia Pacific (Melbourne)
Middle East (Bahrain)	Middle East (Bahrain)
Middle East (UAE)	Middle East (UAE)
South America (São Paulo)	South America (São Paulo)
Israel (Tel Aviv)	Israel (Tel Aviv)
Africa (Cape Town)	Africa (Cape Town)

AWS End User Messaging SMS direct path Region configuration

AWS End User Messaging SMS is available in its own set of AWS Regions, which can differ from the Regions where Amazon SNS or Amazon Cognito user pools are available. For the current list, see Supported Regions and countries in the *AWS End User Messaging SMS User Guide*.

The `Region` field of the `EumsSms` object in SmsConfigurationType sets the AWS Region where Amazon Cognito calls AWS End User Messaging SMS. AWS End User Messaging SMS must be available in your user pool's Region. Omit `Region` and Amazon Cognito uses AWS End User Messaging SMS in your user pool's Region, or set `Region` to your user pool's Region explicitly. Amazon Cognito rejects any other value.

> **Note**
>
> If you set `Region` to an unsupported value, the `CreateUserPool`, `UpdateUserPool`, and `SetUserPoolMfaConfig` requests fail with an `InvalidParameterException`.

> **Note**
>
> The AWS End User Messaging SMS direct path is not available in the Asia Pacific (Thailand), Asia Pacific (Hong Kong), and Asia Pacific (Malaysia) AWS Regions. To send SMS messages from a user pool in one of these Regions, use the Amazon SNS path instead. For more information, see Amazon SNS path Region configuration.

Obtain an origination identity to send SMS messages to US phone numbers

If you plan to send SMS text messages to US phone numbers, you must obtain an origination identity, regardless of whether you build an SMS sandbox testing environment, or a production environment.

US carriers require an origination identity to send messages to US phone numbers. If you don't already have an origination identity, you must get one. To learn how to obtain an origination identity, see Request a phone number in the *AWS End User Messaging SMS User Guide*.

When you have more than one origination identity in the same AWS Region, AWS End User Messaging SMS chooses an origination identity type in the following order of priority: short code, 10DLC, toll-free number. You can't change this priority. For more information, see AWS End User Messaging SMS FAQs.

Confirm that you are in the SMS sandbox

Use the following procedure to confirm that you are in the SMS sandbox. Repeat for each AWS Region where you have production Amazon Cognito user pools.

Review SMS sandbox status in the Amazon Cognito console

To confirm that you are in the SMS sandbox

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.

2. Choose **User Pools**.
3. Choose an existing user pool from the list.
4. Choose the **Authentication methods** menu.
5. In the **SMS configuration** section, expand **Move to Amazon SNS production environment**. If your account is in the SMS sandbox, you will see the following message:

   Configure AWS service dependencies to complete your SMS message setup

   If you don't see this message, then someone has set up SMS messages in your account already. Skip to Complete user pool setup in Amazon Cognito.
6. Choose the Amazon SNS link under **Move to Amazon SNS production environment**. This opens the Amazon SNS console in a new tab.
7. Verify that you are in the sandbox environment. The console message indicates your sandbox status and AWS Region, as follows:

   ```
   This account is in the SMS sandbox in US East (N. Virginia).
   ```

Move your account out of the sandbox

To use your app in production, move your account out of the SMS sandbox and into production. After you have configured an origination identity in the AWS Region that contains the AWS End User Messaging SMS resources that you want Amazon Cognito to use, you can verify US phone numbers while your AWS account remains in the SMS sandbox. When your environment is in production, you don't need to verify user phone numbers before you send SMS messages to them.

You can create a request to exit the sandbox from either the AWS End User Messaging SMS console or the Amazon SNS console. For detailed instructions, see Moving from the SMS Sandbox in the *AWS End User Messaging SMS User Guide*.

Use simulator numbers or verified phone numbers with AWS End User Messaging SMS

If you have moved your account out of the SMS sandbox, skip this step.

If you're in the sandbox but you have set up an origination number, you can send messages to verified destination numbers. To set up verified destinations, see Add a verified destination phone number in the *AWS End User Messaging SMS User Guide*.

You can also send messages with simulated senders and destinations. Simulator messages produce logs but don't get sent out over the carrier network. From the Shortcuts menu, select **Test SMS**

sending with SMS simulator. For more information, see Simulator phone numbers in the *AWS End User Messaging SMS User Guide*.

Complete user pool setup in Amazon Cognito

Return to the browser tab where you were creating or editing your user pool. Complete the procedure. When you have successfully added SMS configuration to your user pool, Amazon Cognito sends a test message to an internal phone number to verify that your configuration works. AWS End User Messaging SMS charges for each test SMS message.

Using Amazon Pinpoint for user pool analytics

> **Note**
>
> **End of support notice:** On October 30, 2026, AWS will end support for Amazon Pinpoint. After October 30, 2026, you will no longer be able to access the Amazon Pinpoint console or Amazon Pinpoint resources (endpoints, segments, campaigns, journeys, and analytics). For more information, see Amazon Pinpoint end of support. **Note:** APIs related to SMS, voice, mobile push, OTP, and phone number validate are not impacted by this change and are supported by AWS End User Messaging.

Amazon Cognito user pools are integrated with Amazon Pinpoint to provide analytics for Amazon Cognito user pools and to enrich the user data for Amazon Pinpoint campaigns. Amazon Pinpoint provides analytics and targeted campaigns to drive user engagement in mobile apps using push notifications. With Amazon Pinpoint analytics support in Amazon Cognito user pools, you can track user pool sign-ups, sign-ins, failed authentications, daily active users (DAUs), and monthly active users (MAUs) in the Amazon Pinpoint console. You can drill into the data for different date ranges or attributes, such as device platform, device locale, and app version.

You can also set up custom attributes for your app. Those can then be used to segment your users on Amazon Pinpoint and send them targeted push notifications. If you choose **Share user attribute data with Amazon Pinpoint** in the **Analytics** configuration for your app client in the **App clients** menu in the Amazon Cognito console, Amazon Pinpoint creates additional endpoints for user email addresses and phone numbers.

When you activate Amazon Pinpoint analytics in your user pool with the Amazon Cognito console, you also create a service-linked role that Amazon Cognito assumes when it makes an API request to Amazon Pinpoint for your user pool. The IAM principal that adds your analytics

configuration must have CreateServiceLinkedRole permissions. The service-linked role is AWSServiceRoleForAmazonCognitoIdp. For more information, see Using service-linked roles for Amazon Cognito.

When you apply an `AnalyticsConfiguration` to your app client in the Amazon Cognito API, you can assign a custom IAM role for Amazon Pinpoint and an external ID to assume the role. The role must trust the `cognito-idp` service principal, and if the role trust policy requires an external ID, it must match your `AnalyticsConfiguration`. You must grant the role `cognito-idp:Describe*` permissions, and the following permissions for your **Amazon Pinpoint project**.

- `mobiletargeting:UpdateEndpoint`
- `mobiletargeting:PutEvents`

Amazon Cognito and Amazon Pinpoint Region availability

The following table shows the AWS Region mappings between Amazon Cognito and Amazon Pinpoint that meet one of the following conditions.

- You can only use an Amazon Pinpoint project in the US East (N. Virginia) (us-east-1) Region.
- You can use an Amazon Pinpoint project in the same Region *or* in the US East (N. Virginia) (us-east-1) Region

By default, Amazon Cognito can only send analytics to a Amazon Pinpoint project in the same AWS Region. The exceptions to this rule are the Regions in the following table, and Regions where Amazon Pinpoint in unavailable.

Amazon Pinpoint isn't available in the following Regions. Amazon Cognito user pools in these Regions don't support analytics.

- Europe (Milan)
- Middle East (Bahrain)
- Asia Pacific (Osaka)
- Israel (Tel Aviv)
- Africa (Cape Town)
- Asia Pacific (Jakarta)
- Asia Pacific (Malaysia)

The table shows the relation between the Region where you built your Amazon Cognito user pool and the corresponding Region in Amazon Pinpoint. You must configure your Amazon Pinpoint project in an available Region to integrate it with Amazon Cognito.

Amazon Cognito user pool Region	Region for Amazon Pinpoint project
ap-northeast-1	us-east-1
ap-northeast-2	us-east-1
ap-south-1	us-east-1, ap-south-1
ap-southeast-1	us-east-1
ap-southeast-2	us-east-1, ap-southeast-2
ca-central-1	us-east-1
eu-central-1	us-east-1, eu-central-1
eu-west-1	us-east-1, eu-west-1
eu-west-2	us-east-1
us-east-1	us-east-1
us-east-2	us-east-1
us-west-2	us-east-1, us-west-2

Region mapping examples

- If you create a user pool in ap-northeast-1, you can create your Amazon Pinpoint project in us-east-1.
- If you create a user pool in ap-south-1, you can create your Amazon Pinpoint project in either us-east-1 or ap-south-1.

> **Note**
>
> For all AWS Regions except those in the preceding table, Amazon Cognito can only use an Amazon Pinpoint project in the same Region as your user pool. If Amazon Pinpoint isn't available in the Region where you built your user pool, and it's not listed in the table, then Amazon Cognito doesn't support Amazon Pinpoint analytics in that Region. For detailed AWS Region information, see Amazon Pinpoint endpoints and quotas.

Specifying Amazon Pinpoint analytics settings (AWS Management Console)

You can configure your Amazon Cognito user pool to send analytics data to Amazon Pinpoint. Amazon Cognito only sends analytics data to Amazon Pinpoint for local users. After you configure your user pool to associate with a Amazon Pinpoint project, you must include `AnalyticsMetadata` in your API requests. For more information, see Integrating your app with Amazon Pinpoint.

To specify analytics settings

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Select **User Pools** and choose an existing user pool from the list.
3. Choose the **App clients** menu and select the app client that you want to update.
4. In the **Analytics** tab under **Pinpoint analytics**, choose **Enable**.
5. Choose a **Pinpoint Region**.
6. Choose an **Amazon Pinpoint project** or select **Create Amazon Pinpoint project**.

> **Note**
>
> The Amazon Pinpoint project ID is a 32-character string that is unique to your Amazon Pinpoint project. It is listed in the Amazon Pinpoint console.
> You can map multiple Amazon Cognito apps to a single Amazon Pinpoint project. However, each Amazon Cognito app can only be mapped to one Amazon Pinpoint project.
> In Amazon Pinpoint, each project should be a single app. For example, if a game developer has two games, each game should be a separate Amazon Pinpoint project,

even if both games use the same Amazon Cognito user pool. For more information on Amazon Pinpoint projects, see Create a project in Amazon Pinpoint.

7. Under **User data sharing**, choose **Share user data with Amazon Pinpoint** if you want Amazon Cognito to send email addresses and phone numbers to Amazon Pinpoint and create additional endpoints for users. After your users verify their email address and phone number, Amazon Cognito only shares them with Amazon Pinpoint if they are available to the user account.

   Note

   An *endpoint* uniquely identifies a user device to which you can send push notifications with Amazon Pinpoint. For more information about endpoints, see Adding endpoints in the *Amazon Pinpoint Developer Guide*.

8. Choose **Save changes**.

Specifying Amazon Pinpoint analytics settings (AWS CLI and AWS API)

Use the following commands to specify Amazon Pinpoint analytics settings for your user pool.

To specify the analytics settings for your user pool's existing client app at app creation time

- AWS CLI: `aws cognito-idp create-user-pool-client`
- AWS API: CreateUserPoolClient

To update the analytics settings for your user pool's existing client app

- AWS CLI: `aws cognito-idp update-user-pool-client`
- AWS API: UpdateUserPoolClient

Note

Amazon Cognito supports in-Region integrations when you use `ApplicationArn`

Integrating your app with Amazon Pinpoint

You can publish analytics metadata to Amazon Pinpoint for Amazon Cognito *local users* in the *user pools API*.

Local users

Users who signed up for an account or were created in your user pool instead of signing in through a third-party identity provider (IdP).

User pools API

The operations that you can integrate with an AWS SDK, using an app with a custom user interface (UI). You can't pass analytics metadata for federated or local users who sign in through managed login. See the Amazon Cognito API Reference for a list of user pools API operations.

After you configure your user pool to publish to a campaign, Amazon Cognito passes metadata to Amazon Pinpoint for the following API operations.

- `AdminInitiateAuth`
- `AdminRespondToAuthChallenge`
- `ConfirmForgotPassword`
- `ConfirmSignUp`
- `ForgotPassword`
- `InitiateAuth`
- `ResendConfirmationCode`
- `RespondToAuthChallenge`
- `SignUp`

To pass metadata about your user's session to your Amazon Pinpoint campaign, include an `AnalyticsEndpointId` value in the `AnalyticsMetadata` parameter of your API request. For a JavaScript example, see Why aren't my Amazon Cognito user pool analytics appearing on my Amazon Pinpoint dashboard? in the *AWS Knowledge Center*.

Protecting user pools with threat protection

You might want to secure your application against network intrusion, password guessing, user impersonation, and malicious sign-up and sign-in. Your configuration of Amazon Cognito user pools security features can be a key component in your security architecture. The security of your application is *Customer responsibility "Security in the cloud"* as described in the AWS Shared Responsibility Model. The tools in this chapter contribute to the ability of your application security design to be in line with these goals.

For account-level and service-wide security guidance—the shared responsibility model, data protection, compliance, and infrastructure security—see Security in Amazon Cognito.

An important decision that you must make when you configure your user pool is whether to permit public sign-up and sign-in. Some user pool option like confidential clients, administrative creation and confirmation of users, and user pools without a domain, are subject to a smaller degree to attacks over the internet. However, a common use case is public clients that accept sign-up from anyone on the internet and send all operations directly to your user pool. In any configuration, but especially in the case of these public configurations, we recommend that you plan and deploy your user pool with security features in mind. Insufficient security can also affect your AWS bill when unwanted sources create new active users or attempt to exploit existing users.

MFA and threat protection apply to local users. Third-party IdPs are responsible for the security posture of federated users.

User pools security features

Multi-factor authentication (MFA)

Request a code that your user pool send by email (with the Essentials or Plus feature plan) or SMS message, or from an authenticator app, to confirm user pool sign-in.

Threat protection

Monitor sign-in for indicators of risk and apply MFA or block sign-in. Add custom claims and scopes to access tokens. Send MFA codes by email.

AWS WAF web ACLs

Inspect incoming traffic to your user pool endpoints and authentication API for unwanted activity at the network and application layers.

Case sensitvity

Prevent creation of users whose email address or preferred username is identical to another user except for character case.

Deletion protection

Prevent automated systems from accidentally deleting your user pools. Require additional confirmation of user pool deletion in the AWS Management Console.

User existence errors

Guard against disclosure of existing usernames and aliases in your user pool. Return a generic error in response to unsuccessful authentication, whether the username is valid or not.

Topics

- Security best practices for Amazon Cognito user pools
- Adding MFA to a user pool
- Authentication levels with ACR and AMR claims
- Working with user devices in your user pool
- Advanced security with threat protection
- Associate an AWS WAF web ACL with a user pool
- User pool case sensitivity
- User pool deletion protection
- Managing user existence error responses

Security best practices for Amazon Cognito user pools

This page describes security best practices that you can implement when you want to guard against common threats. The configuration that you choose will depend on the use case of each application. We recommend that at a minimum, you apply least privilege to administrative operations and take action to guard application and user secrets. Another advanced but effective step that you can take is to configure and apply AWS WAF web ACLs to your user pools.

Protect your user pool at the network level

AWS WAF web ACLs can protect the performance and cost of the authentication mechanisms that you build with Amazon Cognito. With web ACLs, you can implement guardrails in front of

API and managed login requests. Web ACLs create network- and application-layer filters that can drop traffic or require a CAPTCHA based on rules that you devise. Requests aren't passed to your Amazon Cognito resources until they meet the qualifications in your web ACL rules. For more information, see AWS WAF web ACLs.

Protect against SMS message abuse

When you permit public sign-up in your user pool, you can configure account verification with codes that Amazon Cognito sends in SMS text messages. SMS messages can be associated with unwanted activity and increase your AWS bill. Configure your infrastructure to be resilient against sending SMS messages under circumstances of fraud. For more information, review the following posts from AWS Blogs.

- Reduce risks of user sign-up fraud and SMS pumping with Amazon Cognito user pools
- Defending Against SMS Pumping: New AWS Features to Help Combat Artificially Inflated Traffic

Understand public authentication

Amazon Cognito user pools have customer identity and access management (CIAM) features that support use cases where members of the general public can sign up for a user account and access your applications. When a user pool permits self-service sign-up, it's open to requests for user accounts from the public internet. Self-service requests come in from API operations like SignUp and InitiateAuth, and from user interaction with managed login. You can configure user pools to mitigate abuse that might come in from public requests, or disable public authentication operations entirely.

The following settings are some of the ways that you can manage public and internal authentication requests in your user pools and app clients.

Examples of user pool settings that affect public user pool access

Setting	Available options	Configured on	Effect on public authentication	Console setting	API operation and parameter
Self-service sign-up	Permit users to sign up for an account or	User pool	Prevent public sign-up	**Sign-up – Self-**	CreateUserPool,

Setting	Available options	Configured on	Effect on public authentication	Console setting	API operation and parameter
	create user accounts as an administrator.			**service sign-up**	UpdateUserPool `AdminCreateUserConfig – AllowAdminCreateUserOnly`
Administrator confirmation	Send confirmation codes to new users or require administrators to confirm them.	User pool	Prevent confirmation of sign-up without administrator action	**Sign-up** – **Cognito-assisted verification and confirmation**	CreateUserPool, UpdateUserPool `AccountRecoverySettings – admin_only`
User disclosure	Deliver "user not found" messages at sign-in and password reset or prevent disclosure.	App client	Guard against guessing sign-in name, email address, or phone numbers	**App clients** – **Prevent user existence errors**	CreateUserPoolClient, UpdateUserPoolClient `PreventUserExistenceErrors`

Setting	Available options	Configured on	Effect on public authentication	Console setting	API operation and parameter
Client secret	Require or don't require a secret hash at sign-up, sign-in, password reset	App client	Guard against authentication requests from unauthorized sources	**App clients – Client secret**	CreateUserPoolClient `GenerateSecret`
Web ACLs	Enable or don't enable a network firewall for authentication requests	User pool	Limit or prevent access based on administrator-defined request characteristics and IP address rules	**AWS WAF – WAF settings**	AssociateWebACL `ResourceArn`
External IdP	Permit sign-in by users in third-party IdPs, the user pool directory, or both	App client	Exclude local users or federated users from sign-up & sign-in.	**App clients – Identity providers**	CreateUserPoolClient, UpdateUserPoolClient `SupportedIdentityProviders`

Setting	Available options	Configured on	Effect on public authentication	Console setting	API operation and parameter
Authorization server	Host or don't host public webpages for authentication	User pool	Turn off public webpages and allow only SDK-based authentication	**Domain**	CreateUserPoolDomain Creation of any user pool domain makes public webpages available.
Threat protection	Enable or disable monitoring for signs of malicious activity or unsafe passwords	User pool or app client	Can automatically block sign-in or require MFA when users show indicators of compromise	**Threat protection – Protection settings**	SetRiskConfiguration The parameters of `SetRiskConfiguration` define your threat protection settings.

Protect confidential clients with client secrets

The client secret is an optional string that's associated with an app client. All authentication requests to app clients with client secrets must include a secret hash that's generated from the username, client ID, and client secret. Those who don't know the client secret are shut out of your application from the beginning.

However, client secrets have limitations. If you embed a client secret in public client software, your client secret is open to inspection. This opens the ability to create users, submit password-reset requests, and perform other operations in your app client. Client secrets must be implemented only when an application is the only entity that has access to the secret. Typically, this is possible in server-side confidential client applications. This is also true of M2M applications, where a client secret is required. Store the client secret in encrypted local storage or AWS Secrets Manager. Never let your client secret be visible on the public internet.

Protect other secrets

You authentication system with Amazon Cognito user pools might handle private data, passwords, and AWS credentials. The following are some best practices for handling secrets that your application might access.

Passwords

Users might enter passwords when they sign in to your application. Amazon Cognito has refresh tokens that your application can employ to continue expired user sessions without a new password prompt. Don't place any passwords or password hashes in local storage. Design your application to treat passwords as opaque and only pass them through to your user pool.

As a best practice, implement passwordless authentication with WebAuthn passkeys. If you must implement passwords, use the Secure Remote Password (SRP) authentication flow and multi-factor authentication (MFA).

AWS credentials

Administrative authentication and user pool administrative operations require authentication with AWS credentials. To implement these operations in an application, grant secure access to temporary AWS credentials. Grant credentials access only to applications that run on a server component that you control. Don't put applications that have AWS credentials in them on public version-control systems like GitHub. Don't encode AWS credentials in public client-side applications.

PKCE code verifier

Proof Key for Code Exchange, or PKCE, is for OpenID Connect (OIDC) authorization-code grants with your user pool authorization server. Applications share code verifier secrets with your user pool when they request authorization codes. To exchange authorization codes for tokens, clients must reaffirm that they know the code verifier. This practice guards against issuing tokens with intercepted authorization codes.

Clients must generate a new random code verifier with each authorization request. The use of a static or predictable code verifier means that an attacker is only then required to intercept the hardcoded verifier and the authorization code. Design your application so that it doesn't expose code verifier values to users.

User pool administration least privilege

IAM policies can define the level of access that principals have to Amazon Cognito user pool administration and administrative authentication operations. For example:

- To a webserver, grant permissions for authentication with administrative API operations.
- To an AWS IAM Identity Center user who manages a user pool in your AWS account, grant permissions for user pool maintenance and reporting.

The level of resource granularity in Amazon Cognito is limited to two resource types for IAM policy purposes: user pool and identity pool. Note that you can't apply permissions to manage individual app clients. Configure user pools with the knowledge that permissions that you grant are effective across all app clients. When your organization has multiple application tenants and your security model requires separation of administrative responsibilities between tenants, implement multi-tenancy with one tenant per user pool.

Although you can create IAM policies with permissions for user authentication operations like `InitiateAuth`, those permissions have no effect. Public and token-authorized API operations aren't subject to IAM permissions. Of the available user pool authentication operations, you can only grant permissions to *administrative* server-side operations like `AdminInitiateAuth`.

You can limit levels of user pool administration with least-privilege `Action` lists. The following example policy is for an administrator who can manage IdPs, resource servers, app clients, and the user pool domain, but not users or the user pool.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "UserPoolClientAdministrator",
      "Action": [
        "cognito-idp:CreateIdentityProvider",
        "cognito-idp:CreateManagedLoginBranding",
        "cognito-idp:CreateResourceServer",
        "cognito-idp:CreateUserPoolDomain",
        "cognito-idp:DeleteIdentityProvider",
        "cognito-idp:DeleteResourceServer",
        "cognito-idp:DeleteUserPoolDomain",
        "cognito-idp:DescribeIdentityProvider",
        "cognito-idp:DescribeManagedLoginBranding",
        "cognito-idp:DescribeManagedLoginBrandingByClient",
        "cognito-idp:DescribeResourceServer",
        "cognito-idp:DescribeUserPool",
        "cognito-idp:DescribeUserPoolClient",
        "cognito-idp:DescribeUserPoolDomain",
        "cognito-idp:GetIdentityProviderByIdentifier",
        "cognito-idp:GetUICustomization",
        "cognito-idp:ListIdentityProviders",
        "cognito-idp:ListResourceServers",
        "cognito-idp:ListUserPoolClients",
        "cognito-idp:ListUserPools",
        "cognito-idp:SetUICustomization",
        "cognito-idp:UpdateIdentityProvider",
        "cognito-idp:UpdateManagedLoginBranding",
        "cognito-idp:UpdateResourceServer",
        "cognito-idp:UpdateUserPoolClient",
        "cognito-idp:UpdateUserPoolDomain"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws-eusc:cognito-idp:us-west-2:123456789012:userpool/us-
west-2_EXAMPLE"
    }
  ]
}
```

The following example policy grants user and group management and authentication to a server-side application.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "UserAdminAuthN",
      "Action": [
        "cognito-idp:AdminAddUserToGroup",
        "cognito-idp:AdminConfirmSignUp",
        "cognito-idp:AdminCreateUser",
        "cognito-idp:AdminDeleteUser",
        "cognito-idp:AdminDeleteUserAttributes",
        "cognito-idp:AdminDisableProviderForUser",
        "cognito-idp:AdminDisableUser",
        "cognito-idp:AdminEnableUser",
        "cognito-idp:AdminForgetDevice",
        "cognito-idp:AdminGetDevice",
        "cognito-idp:AdminGetUser",
        "cognito-idp:AdminInitiateAuth",
        "cognito-idp:AdminLinkProviderForUser",
        "cognito-idp:AdminListDevices",
        "cognito-idp:AdminListGroupsForUser",
        "cognito-idp:AdminListUserAuthEvents",
        "cognito-idp:AdminRemoveUserFromGroup",
        "cognito-idp:AdminResetUserPassword",
        "cognito-idp:AdminRespondToAuthChallenge",
        "cognito-idp:AdminSetUserMFAPreference",
        "cognito-idp:AdminSetUserPassword",
        "cognito-idp:AdminSetUserSettings",
        "cognito-idp:AdminUpdateAuthEventFeedback",
        "cognito-idp:AdminUpdateDeviceStatus",
        "cognito-idp:AdminUpdateUserAttributes",
        "cognito-idp:AdminUserGlobalSignOut",
        "cognito-idp:AssociateSoftwareToken",
        "cognito-idp:ListGroups",
        "cognito-idp:ListUsers",
        "cognito-idp:ListUsersInGroup",
        "cognito-idp:RevokeToken",
        "cognito-idp:UpdateGroup",
```

```
        "cognito-idp:VerifySoftwareToken"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws-eusc:cognito-idp:us-west-2:123456789012:userpool/us-
west-2_EXAMPLE"
    }
  ]
}
```

Secure and verify tokens

Tokens can contain internal references to group membership and user attributes that you might not want to disclose to the end user. Don't store ID and access tokens in local storage. Refresh tokens are encrypted with a key that only your user pool can access, and are opaque to users and applications. Revoke refresh tokens when users sign out or when you determine that persisting a users' session is undesired for security reasons.

Use access tokens to authorize access only to systems that independently verify that the token is valid and unexpired. For verification resources, see Verifying a JSON web token.

Determine the identity providers that you want to trust

When you configure your user pool with SAML or OIDC identity providers (IdPs), your IdPs can create new users, set user attributes, and access your application resources. SAML and OIDC providers are typically used in business-to-business (B2B) or enterprise scenarios where you or your immediate customer controls membership and configuration of the provider.

Social providers offer user accounts to anyone on the internet and are less under your control than enterprise providers. Only activate social IdPs in your app client when you're ready to allow public customers to sign in and access resources in your application.

Understand the effect of scopes on access to user profiles

You can request access-control scopes in your authentication requests to the user pool authorization server. These scopes can grant your users access to external resources, and they can grant users access to view and modify their own user profiles. Configure your app clients to support the minimum scopes necessary for the operation of your application.

The `aws.cognito.signin.user.admin` scope is present in all access tokens issued by SDK authentication with operations like InitiateAuth. It's designated for user profile self-service

operations in your application. You can also request this scope from your authorization server. This scope is required for token-authorized operations like UpdateUserAttributes and GetUser. The effect of these operations is limited by the read and write permissions of your app client.

The `openid`, `profile`, `email`, and phone scopes authorize requests to the userInfo endpoint on your user pool authorization server. They define the attributes that the endpoint can return. The `openid` scope, when requested without other scopes, returns all available attributes, but when you request more scopes in the request, the response is narrowed down to the attributes represented by the additional scopes. The `openid` scope also indicates a request for an ID token; when you omit this scope from your request to your Authorize endpoint, Amazon Cognito only issues an access token and, when applicable, a refresh token. For more information, see **OpenID Connect scopes** at App client terms.

Sanitize inputs for user attributes

User attributes that might end up as delivery methods and usernames, for example `email`, have format restrictions. Other attributes can have string, boolean, or number data types. String attribute values support a variety of inputs. Configure your application to guard against attempts to write unwanted data to your user directory or the messages that Amazon Cognito delivers to users. Perform client-side validation of user-submitted string attribute values in your application before submitting them to Amazon Cognito.

User pools map attributes from IdPs to your user pool based on an attribute mapping that you specify. Only map secure and predictable IdP attributes to user pool string attributes.

Adding MFA to a user pool

MFA adds a *something you have* authentication factor to the initial *something you know* factor that is typically a username and password. You can choose SMS text messages, email messages, or time-based one-time passwords (TOTP) as additional factors to sign in your users who have passwords as their primary authentication factor.

Multi-factor authentication (MFA) increases security for the local users in your application. In the case of federated users, Amazon Cognito delegates all authentication processes to the IdP and doesn't offer them additional authentication factors.

> **Note**
>
> The first time that a new user signs in to your app, Amazon Cognito issues OAuth 2.0 tokens, even if your user pool requires MFA. The second authentication factor when your user signs in for the first time is their confirmation of the verification message that Amazon Cognito sends to them. If your user pool requires MFA, Amazon Cognito prompts your user to register an additional sign-in factor to use during each sign-in attempt after the first.

With adaptive authentication, you can configure your user pool to require an additional authentication factor in response to an increased risk level. To add adaptive authentication to your user pool, see Advanced security with threat protection.

When you set MFA to `required` for a user pool, all users must complete MFA to sign in. To sign in, each user must set up at least one MFA factor. When MFA is required, you must include the MFA setup in user onboarding so that your user pool permits them to sign in.

Managed login prompts users to set up MFA when you set MFA to be required. When you set MFA to be optional in your user pool, managed login doesn't prompt users. To work with optional MFA, you must build an interface in your app that prompts your users to select that they want to set up MFA, then guides them through the API inputs to verify their additional sign-in factor.

Topics

- Things to know about user pool MFA
- User MFA preferences
- Details of MFA logic at user runtime
- Configure a user pool for multi-factor authentication
- SMS and email message MFA
- TOTP software token MFA

Things to know about user pool MFA

Before you set up MFA, consider the following:

- Users can either have MFA *or* sign in with passwordless factors, with one exception: passkeys with user verification can satisfy MFA requirements as first-factor when you set

`FactorConfiguration` to `MULTI_FACTOR_WITH_USER_VERIFICATION` in your user pool `WebAuthnConfiguration`. Passkeys cannot be used as a second factor to password sign-in.

- You can't set MFA to required in user pools that support one-time passwords.
- You can't add `EMAIL_OTP` or `SMS_OTP` to `AllowedFirstAuthFactors` when MFA is required in your user pool. You can add `WEB_AUTHN` when `FactorConfiguration` is set to `MULTI_FACTOR_WITH_USER_VERIFICATION`.
- Choice-based sign-in only offers `PASSWORD` and `PASSWORD_SRP` factors in all app clients when MFA is required in the user pool. For more information about username-password flows, see Sign-in with persistent passwords and Sign-in with persistent passwords and secure payload in the **Authentication** chapter of this guide.
- In user pools where MFA is optional, users who have configured an MFA factor can only sign in with username-password authentication flows in choice-based sign-in. These users are eligible for all client-based sign-in flows.

The following table describes the effect of user pool MFA settings and user configuration of MFA factors on users' ability to sign in with passwordless factors.

User pool MFA setting	User MFA status	Webauthn/OTP available	Prompted for MFA after password sign-in	Can sign in with WebAuthn/OTP
Required	Configured	No	Yes	No
Required	Not configured	No	No (can't sign in)	No
Optional	Configured	Can set up WebAuthn but can't sign in with passkey	Yes	No
Optional	Not configured	Yes	No	Yes

User pool MFA setting	User MFA status	Webauthn/OTP available	Prompted for MFA after password sign-in	Can sign in with WebAuthn/OTP
Optional (with passkey MFA enabled)	Passkey MFA configured	Yes	Yes (passkey with user verification satisfies MFA independently)	Yes (passkey with user verification satisfies MFA independently)
Required (with passkey MFA enabled)	Passkey MFA configured	Yes	Yes (passkey with user verification satisfies MFA independently)	Yes (passkey with user verification satisfies MFA independently)
Off	Any	Yes	No	Yes

- A user's preferred MFA method influences the methods they can use to recover their password. Users whose preferred MFA is by email message can't receive a password-reset code by email. Users whose preferred MFA is by SMS message can't receive a password-reset code by SMS.

  Your password recovery settings must provide an alternative option when users aren't eligible for your preferred password-reset method. For example, your recovery mechanisms might have email as first priority and email MFA might be an option in your user pool. In this case, add SMS-message account recovery as a second option or use administrative API operations to reset passwords for those users.

  Amazon Cognito replies to password-reset requests from users who don't have a valid recovery method with an `InvalidParameterException` error response.

  The example request body for UpdateUserPool illustrates an `AccountRecoverySetting` where users can fall back to recovery by SMS message when email-message password reset is unavailable.
- Users can't receive MFA and password reset codes at the same email address or phone number. If they use one-time passwords (OTPs) from email messages for MFA, they must use SMS messages for account recovery. If they use OTPs from SMS messages for MFA, they must use

email messages for account recovery. In user pools with MFA, users might be unable to complete self-service password recovery if they have attributes for their email address but no phone number, or their phone number but no email address.

To prevent the state where users can't reset their passwords in user pools with this configuration, set the `email` and `phone_number` attributes as required. As an alternative, you can set up processes that always collect and set those attributes when users sign up or when your administrators create user profiles. When users have both attributes, Amazon Cognito automatically sends password-reset codes to the destination that is *not* the user's MFA factor.

- When you activate MFA in your user pool and choose **SMS message** or **Email message** as a second factor, you can send messages to a phone number or email attribute that you haven't verified in Amazon Cognito. After your user completes MFA, Amazon Cognito sets their `phone_number_verified` or `email_verified` attribute to `true`.
- After five unsuccessful attempts to present an MFA code, Amazon Cognito begins the exponential-timeout lockout process described at Lockout behavior for failed sign-in attempts.
- If your account is in the SMS sandbox in the AWS Region that contains the Amazon Simple Notification Service (Amazon SNS) resources for your user pool, you must verify phone numbers in Amazon SNS before you can send an SMS message. For more information, see SMS message settings for Amazon Cognito user pools.
- To change the MFA status of users in response to detected events with threat protection, activate MFA and set it as optional in the Amazon Cognito user pool console. For more information, see Advanced security with threat protection.
- Email and SMS messages require that your users have email address and phone number attributes respectively. You can set `email` or `phone_number` as required attributes in your user pool. In this case, users can't complete sign-up unless they provide a phone number. If you don't set these attributes as required but want to do email or SMS message MFA, you prompt users for their email address or phone number when they sign up. As a best practice, configure your user pool to automatically message users to verify these attributes.

  Amazon Cognito counts a phone number or email address as verified if a user has successfully received a temporary code by SMS or email message and returned that code in a VerifyUserAttribute API request. As an alternative, your team can set phone numbers and mark them as verified with an administrative application that performs AdminUpdateUserAttributes API requests.
- If you have set MFA to be required and you activated more than one authentication factor, Amazon Cognito prompts new users to select an MFA factor that they want to use. Users

must have a phone number to set up SMS message MFA, and an email address to set up email message MFA. If a user doesn't have the attribute defined for any available message-based MFA, Amazon Cognito prompts them to set up TOTP MFA. The prompt to choose an MFA factor (`SELECT_MFA_TYPE`) and to set up a chosen factor (`MFA_SETUP`) comes in as a challenge response to InitiateAuth and AdminInitiateAuth API operations.

User MFA preferences

Users can set up multiple MFA factors. Only one can be active. You can choose the effective MFA preference for your users in user pool settings or from user prompts. A user pools prompts a user for MFA codes when user pool settings and their own user-level settings meet the following conditions:

1. You set MFA to optional or required in your user pool.
2. The user has a valid `email` or `phone_number`attribute, or has set up an authenticator app for TOTP.
3. At least one MFA factor is active.
4. One MFA factor is set as preferred.

Prevent the use of the same factor for sign-in and MFA

It's possible to configure your user pool in a way that makes one sign-in factor the only available sign-in and MFA option for some or all users. This result can happen when your primary sign-in use case is email-message or SMS-message one-time passwords (OTPs). A user's preferred MFA might be the same type of factor as their sign-in under the following conditions:

- MFA is required in the user pool.
- Email and SMS OTP are available sign-in *and* MFA options in the user pool.
- The user signs in with email or SMS message OTP.
- They have an email-address attribute but no phone-number attribute, or a phone-number attribute but no email-address attribute.

In this scenario, the user can sign in with an email OTP and complete MFA with an email OTP. This option cancels out the essential function of MFA. Users who sign in with one-time passwords must be able to use different delivery methods for sign-in than for MFA. When users have both SMS and

email options, Amazon Cognito automatically assigns a different factor. For example, when a user signs in with email OTP, their preferred MFA is SMS OTP.

Take the following steps to address same-factor authentication when your user pool supports OTP authentication for both sign-in and MFA.

1. Enable both email and SMS OTP as sign-in factors.
2. Enable both email and SMS OTP as MFA factors.
3. Collect

User pool settings and their effect on MFA options

The configuration of your user pool influences the MFA methods that users can choose. The following are some user pool settings that influence users' ability to set up MFA.

- In the **Multi-factor authentication** configuration in the **Sign-in** menu of the Amazon Cognito console, you can set MFA to optional or required, or turn it off. The API equivalent of this setting is the MfaConfiguration parameter of `CreateUserPool`, `UpdateUserPool`, and `SetUserPoolMfaConfig`.

  Also in the **Multi-factor authentication** configuration, the **MFA methods** setting determines the MFA factors that users can set up. The API equivalent of this setting is the SetUserPoolMfaConfig operation.
- In the **Sign-in** menu, under **User account recovery**, you can configure the way that your user pool sends messages to users who forget their password. A user's MFA method can't have the same MFA delivery method as the user pool delivery method for forgot-password codes. The API parameter for the forgot-password delivery method is the AccountRecoverySetting parameter of `CreateUserPool` and `UpdateUserPool`.

  For example, users can't set up email MFA when your recovery option is **Email only**. This is because you can't enable email MFA and set the recovery option to **Email only** in the same user pool. When you set this option to **Email if available, otherwise SMS**, email is the priority recovery option but your user pool can fall back to SMS message when a user isn't eligible for email-message recovery. In this scenario, users can set email MFA as preferred and can only receive an SMS message when they attempt to reset their password.
- If you set only one MFA method as available, you don't need to manage user MFA preferences.

- An active SMS configuration automatically makes SMS messages an available MFA method in your user pool.

  An active email configuration with your own Amazon SES resources in a user pool, and the Essentials or Plus feature plan, automatically makes email messages an available MFA method in your user pool.
- When you set MFA to required in a user pool, users can't enable or disable any MFA methods. You can only set a preferred method.
- When you set MFA to optional in a user pool, managed login doesn't prompt users to set up MFA, but it does prompt users for an MFA code when they have a preferred MFA method.
- When you activate threat protection and configure adaptive-authentication responses in full-function mode, MFA must be optional in your user pool. One of the response options with adaptive authentication is to require MFA for a user whose sign-in attempt is evaluated to contain a level of risk.

  The **Required attributes** setting in the **Sign-up** menu of the console determines whether users must provide an email address or phone number to sign up in your application. Email and SMS messages become eligible MFA factors when a user has the corresponding attribute. The Schema parameter of `CreateUserPool` sets attributes as required.
- When you set MFA to required in a user pool and a user signs in with managed login, Amazon Cognito prompts them to select an MFA method from the available methods for your user pool. Managed login handles the collection of an email address or phone number and the setup of TOTP. The diagram that follows demonstrates the logic behind the options that Amazon Cognito presents to users.

Configure MFA preferences for users

You can configure MFA preferences for users in a self-service model with access-token authorization, or in an administrator-managed model with administrative API operations. These operations enable or disable MFA methods and set one of multiple methods as the preferred option. After your user has set an MFA preference, Amazon Cognito prompts them at sign-in to provide a code from their preferred MFA method. Users who have not set a preference receive a prompt to choose a preferred method in a `SELECT_MFA_TYPE` challenge.

- In a user self-service model or public application, SetUserMfaPreference, authorized with a signed-in user's access token, sets MFA configuration.

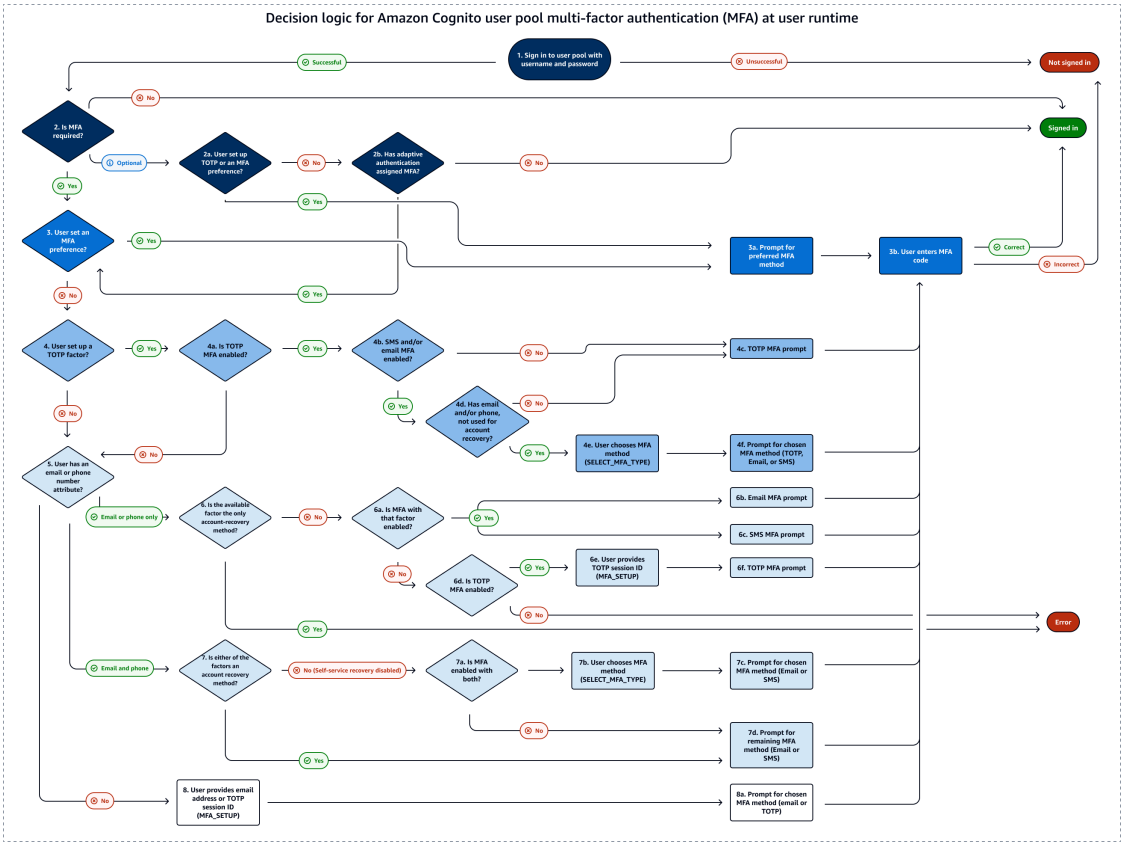
- In an administrator-managed or confidential application, AdminSetUserPreference, authorized with administrative AWS credentials, sets MFA configuration.

You can also set user MFA preferences from the **Users** menu of the Amazon Cognito console. For more information about the public and confidential authentication models in the Amazon Cognito user pools API, see Understanding API, OIDC, and managed login pages authentication.

Details of MFA logic at user runtime

To determine the steps to take when users sign in, your user pool evaluates user MFA preferences, user attributes, the user pool MFA setting, threat protection actions, and self-service account recovery settings. It then signs users in, prompts them to choose an MFA method, prompts them to set up an MFA method, or prompts them for MFA. To set up an MFA method, users must provide an email address or phone number or register a TOTP authenticator. They can also set up MFA options and register a preferred option in advance. The following diagram lists the detailed effects of user pool configuration on sign-in attempts immediately after initial sign-up.

The logic illustrated here applies to SDK-based applications and managed login sign-in, but is less visible in managed login. When you troubleshoot MFA, work backward from your users' outcomes to the user-profile and user-pool configurations that contributed to the decision.



The following list corresponds to the numbering in the decision logic diagram and describes each step in detail. A

indicates a successful authentication and the conclusion of the flow. A

indicates unsuccessful authentication.

1. A user presents their username or username and password at your sign-in screen. If they don't present valid credentials, their sign-in request is denied.
2. If they succeed username-password authentication, determine whether MFA is required, optional, or off. If it is off, the correct username and password results in successful authentication.

   a. If MFA is optional, determine if the user has previously set up a TOTP authenticator. If they have set up TOTP, prompt for TOTP MFA. If

they successfully respond to the MFA challenge, they're signed in.

b. Determine if the adaptive authentication feature of threat protection has required the user to set up MFA. If it hasn't assigned MFA, the user is signed in.

3. If MFA is required or adaptive authentication has assigned MFA, determine if the user has set an MFA factor as enabled and preferred. If they have, prompt for MFA with that factor. If they successfully respond to the MFA challenge, they're signed in.

4. If the user hasn't set an MFA preference, determine if the user has registered a TOTP authenticator.
   a. If the user has registered a TOTP authenticator, determine if TOTP MFA is available in the user pool (TOTP MFA can be disabled after users have previously set up authenticators).
   b. Determine whether email-message or SMS-message MFA is also available in the user pool.
   c. If neither email nor SMS MFA is available, prompt the user for TOTP MFA. If they successfully respond to the MFA challenge, they're signed in.

   d. If email or SMS MFA are available, determine whether the user has the corresponding `email` or `phone_number` attribute. If so, any attribute that isn't the primary method for self-service account recovery and is enabled for MFA is available to them.
   e. Prompt the user with a `SELECT_MFA_TYPE` challenge with `MFAS_CAN_SELECT` options that include TOTP and the available SMS or email MFA factors.
   f. Prompt the user for the factor that they select in response to the `SELECT_MFA_TYPE` challenge. If they successfully respond to the MFA challenge, they're signed in.

5. If the user hasn't registered a TOTP authenticator, or if they have but TOTP MFA is currently disabled, determine whether the user has an `email` or `phone_number` attribute.
6. If the user has only an email address or only a phone number, determine whether that attribute is also the method the user pool implements to send account-recovery messages for password reset. If true, they can't complete sign-in with MFA required and Amazon Cognito returns an error. To activate sign-in for this user,

you must add a non-recovery attribute or register a TOTP authenticator for them.

a. If they have an available non-recovery email address or phone number, determine whether the corresponding email or SMS MFA factor is enabled.

b. If they have a non-recovery email address attribute and email MFA is enabled, prompt them with an `EMAIL_OTP` challenge. If they successfully respond to the MFA challenge, they're signed in.

c. If they have a non-recovery phone number attribute and SMS MFA is enabled, prompt them with an `SMS_MFA` challenge. If they successfully respond to the MFA challenge, they're signed in.

d. If they don't have an attribute that's eligible for an enabled email or SMS MFA factor, determine whether TOTP MFA is enabled. If TOTP MFA is disabled, they can't complete sign-in with MFA required and Amazon Cognito returns an error. To activate sign-in for this user, you must add a non-recovery attribute or register a TOTP authenticator for them.

> **Note**
>
> This step has already been evaluated as **No** if the user has a TOTP authenticator but TOTP MFA is disabled.

e. If TOTP MFA is enabled, present the user with a `MFA_SETUP` challenge with `SOFTWARE_TOKEN_MFA` in the `MFAS_CAN_SETUP` options. To complete this challenge, you must separately register a TOTP authenticator for the user and respond with `"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "SESSION": "[Session ID from VerifySoftwareToken]}"`.

f. After the user responds to the `MFA_SETUP` challenge with the session token from a VerifySoftwareToken request, prompt them with an `SOFTWARE_TOKEN_MFA` challenge. If they successfully respond to the MFA challenge, they're signed in.

7. If the user has both an email address and phone number, determine which attribute, if any, is the primary method for account-recovery messages for password reset.
   a. If self-service account recovery is disabled, either attribute can be used for MFA. Determine whether one or both of the email and SMS MFA factors are enabled.
   b. If both attributes are enabled as an MFA factor, prompt the user with a `SELECT_MFA_TYPE` challenge with `MFAS_CAN_SELECT` options `SMS_MFA` and `EMAIL_OTP`.
   c. Prompt them for the factor that they select in response to the `SELECT_MFA_TYPE` challenge. If they successfully respond to the MFA challenge, they're signed in.

   d. If only one attribute is an eligible MFA factor, prompt them with a challenge for the remaining factor. If they successfully respond to the MFA challenge, they're signed in.

      This outcome happens in the following scenarios.

      i. When they have `email` and `phone_number` attributes, SMS and email MFA are enabled, and the primary account-recovery method is by email or SMS message.
      ii. When they have `email` and `phone_number` attributes, only SMS MFA or email MFA is enabled, and self-service account recovery is disabled.

8. If the user hasn't registered a TOTP authenticator and has neither an `email` nor `phone_number` attribute, prompt them with an `MFA_SETUP` challenge. The list in `MFAS_CAN_SETUP` includes all enabled MFA factors in the user pool that aren't the primary account-recovery option. They can respond to this challenge with `ChallengeResponses` for email or TOTP MFA. To set up SMS MFA, add a phone number attribute separately and restart authentication.

   For TOTP MFA, respond with `"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "SESSION": "[Session ID from VerifySoftwareToken]"}`.

   For email MFA, respond with `"ChallengeName": "MFA_SETUP", "ChallengeResponses": {"USERNAME": "[username]", "email": "[user's email address]"}`.

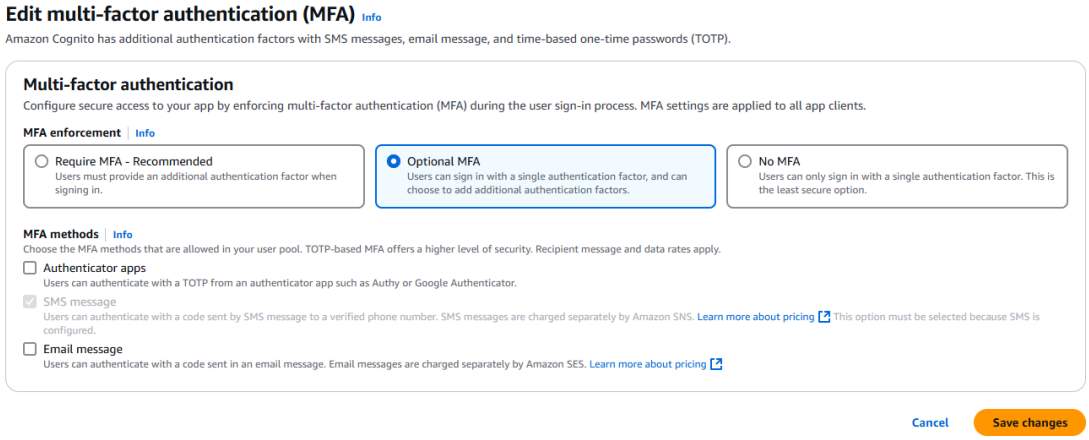
   a. Prompt them for the factor that they select in response to the `SELECT_MFA_TYPE` challenge. If they successfully respond to the MFA challenge, they're signed in.

Configure a user pool for multi-factor authentication

You can configure MFA in the Amazon Cognito console or with the SetUserPoolMfaConfig API operation and SDK methods.

To configure MFA in the Amazon Cognito console

1. Sign in to the Amazon Cognito console.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Sign-in** menu. Locate **Multi-factor authentication** and choose **Edit**.
5. Choose the **MFA enforcement** method that you want to use with your user pool.



   a. **Require MFA**. All users in your user pool must sign in with an additional SMS, email, or time-based one-time password (TOTP) code as an additional authentication factor.

   b. **Optional MFA**. You can give your users the option to register an additional sign-in factor but still permit users who haven't configured MFA to sign in. If you use adaptive authentication, choose this option. For more information about adaptive authentication, see Advanced security with threat protection.

   c. **No MFA**. Your users can't register an additional sign-in factor.

6. Choose the **MFA methods** that you support in your app. You can set **Email message**, **SMS message** or TOTP-generating **Authenticator apps** as a second factor.
7. If you use SMS text messages as a second factor and you haven't configured an IAM role to use with Amazon Simple Notification Service (Amazon SNS) for SMS messages, create one in the console. In the **Authentication methods** menu for your user pool, locate **SMS** and choose **Edit**.

You can also use an existing role that allows Amazon Cognito to send SMS messages to your users for you. For more information, see IAM Roles.

If you use email messages as a second factor and you haven't configured an originating identity to use with Amazon Simple Email Service (Amazon SES) for email messages, create one in the console. You must choose the **Send email with SES** option. In the **Authentication methods** menu for your user pool, locate **Email** and choose **Edit**. Select a **FROM email address** from the available verified identities in the list. If you choose a verified domain, for example `example.com`, you must also configure a **FROM sender name** in the verified domain, for example `admin-noreply@example.com`.

8. Choose **Save changes**.

SMS and email message MFA

SMS and email MFA messages confirm that users have access to a message destination before they can sign in. They confirm that they not only have access to a password, but to the SMS messages or the email inbox of the original user. Amazon Cognito requests that users provide a short code that your user pool sent after they successfully provide a username and password.

SMS and email message MFA require no additional configuration after your user adds an email address or phone number to their profile. Amazon Cognito can send messages to unverified email addresses and phone numbers. When a user completes their first MFA, Amazon Cognito marks their email address or phone number as verified.

MFA authentication begins when a user with MFA enters their username and password in your application. Your application submits these initial parameters in an SDK method that invokes an InitiateAuth or AdminInitiateAuth API request. The `ChallengeParameters` in the API response includes a `CODE_DELIVERY_DESTINATION` value that indicates where the authorization code was sent. In your application, display a form that prompts the user to check their phone and includes an input element for the code. When they enter their code, submit it in a challenge-response API request to complete the sign-in process.

After a user with MFA signs in with username and password in the managed login pages, they're automatically prompted for the MFA code.

User pools send SMS messages for MFA and other Amazon Cognito notifications with Amazon Simple Notification Service (Amazon SNS) resources in your AWS account. Similarly, users pools send email messages with Amazon Simple Email Service (Amazon SES) resources in your account.

These linked services incur their own costs on your AWS bill for message delivery. They also have additional requirements for sending messages at production volumes. See the following links for more information:

- SMS message settings for Amazon Cognito user pools
- Worldwide SMS Pricing
- Email settings for Amazon Cognito user pools
- Amazon SES pricing

Considerations for SMS and email message MFA

- To permit users to sign in with email MFA, your user pool must have the following configuration options:
  1. You have the Plus or Essentials feature plan in your user pool. For more information, see User pool feature plans.
  2. Your user pool sends email messages with your own Amazon SES resources. For more information, see Amazon SES email configuration.
- The MFA code is valid for the **Authentication flow session duration** that you set for your app client.

  Set the duration of an authentication flow session in the Amazon Cognito console in the **App clients** menu when you **Edit** your app client. You can also set the authentication flow session duration in a `CreateUserPoolClient` or `UpdateUserPoolClient` API request. For more information, see An example authentication session.
- When a user successfully provides a code from an SMS or email message that Amazon Cognito sent to an unverified phone number or email address, Amazon Cognito marks the corresponding attribute as verified.
- For a user to make a self-service change to the value of a phone number or email address that's associated with MFA, they must sign in and authorize the request with an access token. If they can't access their current phone number or email address, they can't sign in. Your team must change these values with administrator AWS credentials in AdminUpdateUserAttributes API requests.
- After you configure SMS in your user pool, you can't disable SMS messages as an available MFA factor.

TOTP software token MFA

When you set up TOTP software token MFA in your user pool, your user signs in with a username and password, then uses a TOTP to complete authentication. After your user sets and verifies a username and password, they can activate a TOTP software token for MFA. If your app uses the Amazon Cognito managed login to sign in users, your user submits their username and password, and then submits the TOTP password on an additional sign-in page.

You can activate TOTP MFA for your user pool in the Amazon Cognito console, or you can use Amazon Cognito API operations. At the user pool level, you can call SetUserPoolMfaConfig to configure MFA and enable TOTP MFA.

Note

If you haven't activated TOTP software token MFA for the user pool, Amazon Cognito can't use the token to associate or verify users. In this case, users receive a `SoftwareTokenMFANotFoundException` exception with the description `Software Token MFA has not been enabled by the userPool`. If you deactivate software token MFA for the user pool later, users who previously associated and verified a TOTP token can continue to use it for MFA.

Configuring TOTP for your user is a multi-step process where your user receives a secret code that they validate by entering a one-time password. Next, you can enable TOTP MFA for your user or set TOTP as the preferred MFA method for your user.

When you configure your user pool to require TOTP MFA and your users sign up for your app in managed login, Amazon Cognito automates the user process. Amazon Cognito prompts your user to choose an MFA method, displays a QR code to set up their authenticator app, and verifies their MFA registration. In user pools where you have allowed users to choose between SMS and TOTP MFA, Amazon Cognito also presents your user with a choice of method.

Important

When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login TOTP registration. To create a rule that has a CAPTCHA action and doesn't affect managed login TOTP, see Configuring your AWS WAF web ACL for managed login TOTP MFA. For more

information about AWS WAF web ACLs and Amazon Cognito, see Associate an AWS WAF web ACL with a user pool.

To implement TOTP MFA in a custom-built UI with an AWS SDK and the Amazon Cognito user pools API, see Configuring TOTP MFA for a user.

To add MFA to your user pool, see Adding MFA to a user pool.

TOTP MFA considerations and limitations

1. Amazon Cognito supports software token MFA through an authenticator app that generates TOTP codes. Amazon Cognito doesn't support hardware-based MFA.
2. When your user pool requires TOTP for a user who has not configured it, your user receives a one-time access token that your app can use to activate TOTP MFA for the user. Subsequent sign-in attempts fail until your user has registered an additional TOTP sign-in factor.
   - A user who signs up in your user pool with the `SignUp` API operation or through managed login receives one-time tokens when the user completes sign-up.
   - After you create a user, and the user sets their initial password, Amazon Cognito issues one-time tokens from managed login to the user. If you set a permanent password for the user, Amazon Cognito issues one-time tokens when the user first signs in.
   - Amazon Cognito doesn't issue one-time tokens to an administrator-created user who signs in with the InitiateAuth or AdminInitiateAuth API operations. After your user succeeds in the challenge to set their initial password, or if you set a permanent password for the user, Amazon Cognito immediately challenges the user to set up MFA.
3. If a user in a user pool that requires MFA has already received a one-time access token but hasn't set up TOTP MFA, the user can't sign in with managed login until they have set up MFA. Instead of the access token, you can use the `session` response value from an `MFA_SETUP` challenge to InitiateAuth or AdminInitiateAuth in an AssociateSoftwareToken request.
4. If your users have set up TOTP, they can use it for MFA, even if you deactivate TOTP for the user pool later.
5. Amazon Cognito only accepts TOTPs from authenticator apps that generate codes with the HMAC-SHA1 hash function. Codes generated with SHA-256 hashing return a `Code mismatch` error.

Configuring TOTP MFA for a user

When a user first signs in, your app uses their one-time access token to generate the TOTP private key and present it to your user in text or QR code format. Your user configures their authenticator app and provides a TOTP for subsequent sign-in attempts. Your app or managed login presents the TOTP to Amazon Cognito in MFA challenge responses.

Under some circumstances, managed login prompts new users to set up a TOTP authenticator. for more information, see Details of MFA logic at user runtime.

Topics

- Associate the TOTP software token
- Verify the TOTP token
- Sign in with TOTP MFA
- Remove the TOTP token

Associate the TOTP software token

To associate the TOTP token, send your user a secret code that they must validate with a one-time password. Associating the token requires three steps.

1. When your user chooses TOTP software token MFA, call AssociateSoftwareToken to return a unique generated shared secret key code for the user account. You can authorize AssociateSoftwareToken with either an access token or a session string.
2. Your app presents the user with the private key, or a QR code that you generate from the private key. Your user must enter the key into a TOTP-generating app like Google Authenticator, either by scanning the QR code that your application generates from the private key or by manually entering the key.
3. Your user enters the key, or scans the QR code into a authenticator app such as Google Authenticator, and the app begins generating codes.

Verify the TOTP token

Next, verify the TOTP token. Request sample codes from your user and provide them to the Amazon Cognito service to confirm that the user is successfully generating TOTP codes, as follows.

1. Your app prompts your user for a code to demonstrate that they have set up their authenticator app properly.
2. The user's authenticator app displays a temporary password. The authenticator app bases the password on the secret key you gave to the user.
3. Your user enters their temporary password. Your app passes the temporary password to Amazon Cognito in a `VerifySoftwareToken` API request.
4. Amazon Cognito has retained the secret key associated with the user, and generates a TOTP and compares it with the one that your user provided. If they match, `VerifySoftwareToken` returns a `SUCCESS` response.
5. Amazon Cognito associates the TOTP factor with the user.
6. If the `VerifySoftwareToken` operation returns an `ERROR` response, make sure that the user's clock is correct and that they have not exceeded the maximum number of retries. Amazon Cognito accepts TOTP tokens that are within 30 seconds before or after the attempt, to account for minor clock skew. When you have resolved the issue, try the VerifySoftwareToken operation again.

Sign in with TOTP MFA

At this point, your user signs in with the time-based one-time password. The process is as follows.

1. Your user enters their username and password to sign in to your client app.
2. The TOTP MFA challenge is invoked, and your user is prompted by your app to enter a temporary password.
3. Your user gets the temporary password from an associated TOTP-generating app.
4. Your user enters the TOTP code into your client app. Your app notifies the Amazon Cognito service to verify it. For each sign-in, RespondToAuthChallenge should be called to get a response to the new TOTP authentication challenge.
5. If the token is verified by Amazon Cognito, the sign-in is successful and your user continues with the authentication flow.

Remove the TOTP token

Finally, your app should allow your user to deactivate their TOTP configuration. To delete your user's TOTP software token, use AdminDeleteSoftwareToken. After you delete your user's software token, they can no longer sign in with TOTP MFA and can register a new software token. Use this

operation when your user loses access to their TOTP-generating device, for example a lost or reset phone. When your user pool requires MFA and no other factor is available, Amazon Cognito returns an `MFA_SETUP` challenge at your user's next sign-in. When your user has another factor available, such as SMS message MFA, sign-in falls back to that factor.

If your user still has access to their TOTP-generating device, you don't need to delete their software token. Amazon Cognito disassociates the existing software token when your user verifies a new one. To deactivate TOTP MFA for your user, call SetUserMFAPreference to modify your user to use no MFA, or only SMS MFA.

1. Create an interface in your app for users who want to reset MFA. Prompt a user in this interface to enter their password.
2. If Amazon Cognito returns a TOTP MFA challenge, update your user's MFA preference with SetUserMFAPreference.
3. In your app, communicate to your user that they have deactivated MFA and prompt them to sign in again.

Configuring your AWS WAF web ACL for managed login TOTP MFA

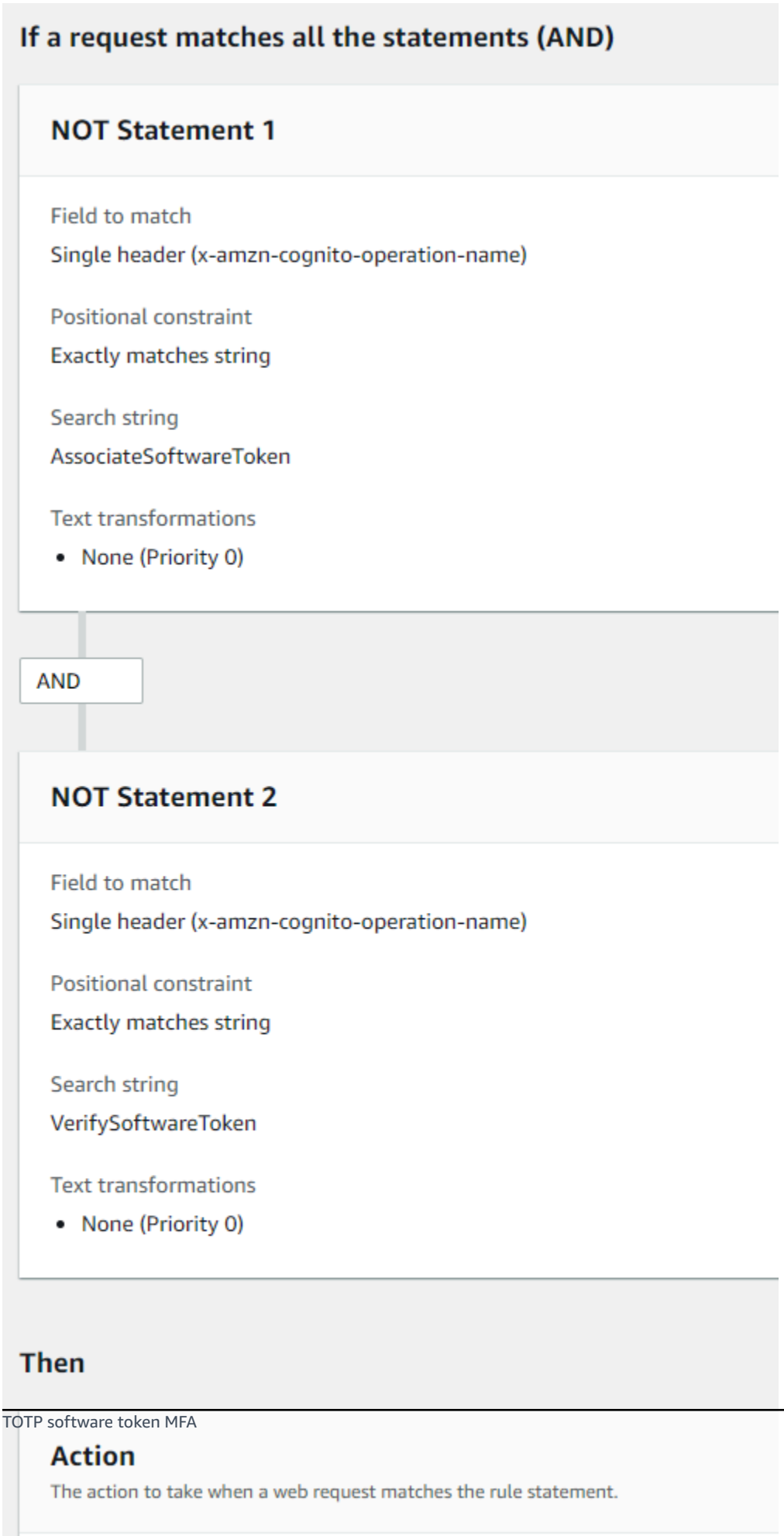
When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login TOTP registration. AWS WAF CAPTCHA rules *only* have this effect on TOTP MFA in managed login and the classic hosted UI. SMS MFA is unaffected.

Amazon Cognito displays the following error when your CAPTCHA rule doesn't let a user complete TOTP MFA setup.

Request not allowed due to WAF captcha.

This error results when AWS WAF prompts for a CAPTCHA in response to AssociateSoftwareToken and VerifySoftwareToken API requests that your user pool makes in the background. To create a rule that has a CAPTCHA action and doesn't affect TOTP in managed login pages, exclude the `x-amzn-cognito-operation-name` header values of `AssociateSoftwareToken` and `VerifySoftwareToken` from the CAPTCHA action in your rule.

The following screenshot shows an example AWS WAF rule that applies a CAPTCHA action to all requests that don't have a `x-amzn-cognito-operation-name` header value of `AssociateSoftwareToken` or `VerifySoftwareToken`.


If a request matches all the statements (AND)
NOT Statement 1
Field to match
Single header (x-amzn-cognito-operation-name)
Positional constraint
Exactly matches string
Search string
AssociateSoftwareToken
Text transformations
None (Priority 0)
AND
NOT Statement 2
Field to match
Single header (x-amzn-cognito-operation-name)
Positional constraint
Exactly matches string
Search string
VerifySoftwareToken
Text transformations
None (Priority 0)
Then
Action
The action to take when a web request matches the rule statement.

For more information about AWS WAF web ACLs and Amazon Cognito, see Associate an AWS WAF web ACL with a user pool.

Authentication levels with ACR and AMR claims

Amazon Cognito can record how strongly a user authenticated and which methods they used. You provide a target authentication level as an input when a user signs in, and Amazon Cognito reports the level that the user reached and the methods that they completed as claims in the tokens that it issues. These claims apply to any sign-in, including a user's initial sign-in.

You request an authentication level with the `acr_values` parameter on the managed login authorize and login endpoints, or with the equivalent `TARGET_ACR_VALUES` parameter in the `USER_AUTH` API flow. Both accept an ordered list of target authentication levels. You can add the optional `max_age` parameter (`MAX_AGE` in the API) to require that the user authenticated within a recent time window. In response to these inputs, Amazon Cognito adds two claims to the tokens that it issues:

- The `acr` claim is a string that indicates the authentication level that the user satisfied, for example `urn:cognito:loa:3`.
- The `amr` claim is an array of strings that indicates the authentication methods that the user completed, for example `["pwd", "otp", "mfa"]`.

The `acr` and `amr` claims follow the definitions in the OpenID Connect Core specification. Amazon Cognito uses its own ACR level URIs. You can translate the Amazon Cognito levels to whatever standard your application requires.

Step-up authentication is a common use case for these claims. In step-up authentication, your application requires a user to authenticate with a stronger method before they perform a sensitive operation. To decide whether to prompt, your application compares the `acr` level and the `auth_time` claim in the user's ID token against the level and freshness that the operation requires, and requests step-up only when they aren't already met. When step-up is needed, redirect the user to the managed login authorize or login endpoint with `acr_values`, or call the `USER_AUTH` API flow with `TARGET_ACR_VALUES`. For an example, see the Step-up authentication scenario.

You can request ACR and AMR from any Amazon Cognito sign-in entry point. The mechanics for each entry point are documented with that entry point.

- SDK and API sign-in — Request a target level in the `USER_AUTH` flow with the `TARGET_ACR_VALUES` and `MAX_AGE` parameters. For the parameters and step-up behavior, see Step-up authentication with the USER_AUTH flow.
- Managed login and hosted UI — Pass the `acr_values` and `max_age` parameters on the authorize and login endpoints. For details, see the authorize endpoint and login endpoint.
- Federated sign-in — Amazon Cognito derives the ACR and AMR values of a federated user from the identity provider (IdP) response and maps them with the `AcrMapping` field. For OIDC identity providers, see Step-up authentication for federated users.

Requesting step-up authentication requires the Essentials or Plus feature plan. For more information, see Feature plan requirements.

Topics

- Authentication levels and methods
- Configuring ACR level names
- Feature plan requirements
- Your responsibilities for step-up authentication

Authentication levels and methods

Amazon Cognito evaluates the authentication factors that a user completes and expresses the result as an authentication level in the `acr` claim and a list of authentication methods in the `amr` claim.

ACR authentication levels

Amazon Cognito defines four fixed authentication levels. The authentication factor combinations that satisfy each level are fixed and you can't change them. You can customize only the URI name of each level for your user pool. For more information, see Configuring ACR level names.

Amazon Cognito assigns the highest authentication level that the user's completed factors fully satisfy.

ACR levels and satisfying factor combinations

ACR level	Default ACR value	Satisfying factor combinations
1	`urn:cognito:loa:1`	Password only
2	`urn:cognito:loa:2`	A single one-time password delivered by SMS or email. Level 2 doesn't include passkeys
3	`urn:cognito:loa:3`	A passkey (platform or roaming) on its own; or a password with an SMS one-time code; or a password with an email one-time code
4	`urn:cognito:loa:4`	A password with a time-based one-time password (TOTP) from an authenticator app

Level 1 represents password-only authentication. Federation doesn't satisfy a level through this table. Instead, Amazon Cognito derives the authentication level of a federated user from the response of the identity provider (IdP). For more information, see Step-up authentication for federated users.

AMR values

The `amr` claim is a list of strings. Each authentication factor that the user completes adds its value to the list. Amazon Cognito can add the following values.

AMR values

Authentication method	AMR value	Notes
Password	`pwd`	Password-family sign-in, including Secure Remote Password (SRP)
TOTP from an authenticator app	`otp`	A time-based one-time password from an app such as an authenticator app
SMS one-time code	`sms`	A one-time code delivered by SMS
Email one-time code	`email_otp`	A one-time code delivered by email

Authentication method	AMR value	Notes
Passkey with a platform authenticator	`swk`	A software-bound key, for example a passkey secured by a device biometric
Passkey with a roaming authenticator	`hwk`	A hardware-bound key, for example a security key
Multi-factor combination	`mfa`	Added when the user completes two or more distinct factors
Federated sign-in	`fed`	A default placeholder that Amazon Cognito adds when a user federates through an IdP that doesn't support ACR and AMR, or returns no values that Amazon Cognito recognizes

ACR and AMR token claims

Amazon Cognito adds the `acr` and `amr` claims to access tokens and ID tokens. Your application inspects the `acr` and `amr` claims in the access token or ID token to evaluate the authentication level. When a user refreshes their tokens, Amazon Cognito preserves the same `acr` and `amr` values on the new access and ID tokens, so the user isn't asked to step up again after every refresh. For more information about token contents, see Using the ID token and Using the access token.

The following example shows an ID token after a user authenticates with a password and a TOTP.

```
{
  "sub": "a1b2c3d4-5678-90ab-cdef-1234567890ab",
  "iss": "https://cognito-idp.us-east-1.amazonaws.com/us-east-1_ExAmPlE",
  "aud": "abc123def456",
  "exp": 1759348800,
  "iat": 1759345200,
  "auth_time": 1759345200,
  "token_use": "id",
  "acr": "urn:cognito:loa:4",
  "amr": ["pwd", "otp", "mfa"],
  "email": "user@example.com",
  "cognito:username": "a1b2c3d4-5678-90ab-cdef-1234567890ab"
}
```

When a user refreshes their tokens, Amazon Cognito sets the same `acr` and `amr` values on the new access and ID tokens. The values carry over without modification. This behavior provides a smoother experience because users aren't asked to step up again after every refresh.

Refreshing a token doesn't reset the `auth_time` claim. If you want to force a user to authenticate again after a period of time, use the `max_age` parameter, which compares the current time to `auth_time` regardless of how many times the user refreshed their tokens.

When a user completes step-up authentication, Amazon Cognito issues new access, ID, and refresh tokens with the updated `acr` and `amr` claims and a fresh validity period.

Configuring ACR level names

You can customize the URI name of each authentication level for your user pool with the `AcrConfiguration` field. Configure this field when you create or update a user pool with the CreateUserPool or UpdateUserPool operation. The factor combinations that satisfy each level stay fixed. You customize only the names.

Configuring custom ACR level names with the `AcrConfiguration` field requires the Essentials or Plus feature plan. For more information, see Feature plan requirements.

The `AcrConfiguration` field is a map with the keys `Level1` through `Level4`. Each value is an object with a single `AcrValue` string. You can override only the levels that you want to customize. The following example request customizes the names of levels 2 and 3.

```
{
  "AcrConfiguration": {
    "Level2": { "AcrValue": "urn:my-standard:loa:2" },
    "Level3": { "AcrValue": "urn:my-standard:loa:3" }
  }
}
```

The response always returns the effective configuration, with the default names merged in for any levels that you didn't customize.

```
{
  "AcrConfiguration": {
    "Level1": { "AcrValue": "urn:cognito:loa:1" },
    "Level2": { "AcrValue": "urn:my-standard:loa:2" },
```

```
    "Level3": { "AcrValue": "urn:my-standard:loa:3" },
    "Level4": { "AcrValue": "urn:cognito:loa:4" }
  }
}
```

The following validation rules apply to each `AcrValue`.

- Each `AcrValue` must be unique across all four levels. This includes the default names that fill in for levels that you don't customize. A value that collides with the default for an uncustomized level is rejected.
- Each `AcrValue` is a string of 1 to 64 characters.
- Each `AcrValue` can't contain spaces, double quotation marks, or backslashes.

If you change your ACR level names, a user might later present an access token whose `acr` value no longer matches any configured level. Amazon Cognito doesn't respect the stale value and treats it as an unrecognized level, but this isn't an error. Amazon Cognito still honors the token's `amr` values and prompts only for factors that aren't already listed in the `amr` claim, without crediting any particular level.

Feature plan requirements

The following capabilities require the Essentials or Plus feature plan.

- Requesting step-up authentication. This applies to the `TARGET_ACR_VALUES` parameter on `InitiateAuth` and the `acr_values` parameter on managed login.
- Configuring custom ACR level names with the `AcrConfiguration` field on `CreateUserPool` or `UpdateUserPool`.

The behavior when you request step-up authentication on an insufficient feature plan depends on the entry point.

- The `InitiateAuth` API returns a `FeatureUnavailableInTierException`.
- The managed login authorize endpoint silently ignores `acr_values` and proceeds with normal authentication, without an error.

For more information about feature plans, see User pool feature plans.

Your responsibilities for step-up authentication

Amazon Cognito issues tokens with the authentication level that the user achieved. Your application is responsible for the following.

- Check the user's current ID token before you redirect. Evaluate `auth_time` against your `max_age`, and the ID token's `acr` level against the level that the operation requires, with the tokens that your application already holds. Redirect the user to Amazon Cognito for step-up only if the required level or freshness isn't already met.
- Set the correct `TARGET_ACR_VALUES` or `acr_values`, and the correct `MAX_AGE` or `max_age`. Amazon Cognito acts only on what you provide. It doesn't infer the level or freshness that you intend. If you request the wrong level or omit the parameters, Amazon Cognito doesn't step the user up as you intended.
- Validate the `acr` claim in your application. After Amazon Cognito returns tokens, your application must verify that the `acr` claim matches the level that the operation requires before it grants access. Amazon Cognito issues the token with the achieved level. Enforcing that an operation requires a given level is the responsibility of your application.
- Translate Amazon Cognito levels to other standards outside the token. Amazon Cognito computes the `acr` and `amr` claims from the authentication flow. You can't customize, override, add to, or suppress these claims with a pre token generation Lambda trigger or any other Lambda trigger. If you need to translate the Amazon Cognito levels to another standard, such as NIST or eIDAS, do so in your application layer.

Working with user devices in your user pool

When you sign in local user pool users with the Amazon Cognito user pools API, you can associate your users' activity logs from threat protection with each of their devices and, optionally, allow your users to skip multi-factor authentication (MFA) if they're on a trusted device. Amazon Cognito includes a device key in the response to any sign-in that doesn't already include device information. The device key is in the format *`Region_UUID`*. With a device key, a Secure Remote Password (SRP) library, and a user pool that permits device authentication, you can prompt users in your app to trust the current device and no longer prompt for an MFA code at sign-in.

Topics

- Setting up remembered devices
- Getting a device key

- Signing in with a device
- Viewing, updating and forgetting devices

Setting up remembered devices

With Amazon Cognito user pools, you can associate each of your users' devices with a unique device identifier: a device key. When you present the device key and perform device authentication at sign-in, you can configure your application with a *trusted device* authentication flow. In this flow, your application can present a choice to users to sign in without MFA until a later time, as determined by the security requirements of your app or the preferences of your users. At the end of that time period, your application must change the device status to *not remembered* and the user must sign in with MFA until they confirm that they want to remember a device. For example, your application might prompt your users to trust a device for 30, 60, or 90 days. You can store this date in a custom attribute and on that date, change the remembered status of their device. You must then re-prompt your user to submit an MFA code and set the device to be remembered again after successful authentication.

1. Remembered devices can override MFA only in user pools with MFA active.

When your user signs in with a remembered device, you must perform an additional device authentication during their authentication flow. For more information, see Signing in with a device.

Configure your user pool to remember devices in the **Sign-in** menu of your user pool, under **Device tracking**. When setting up the remembered devices functionality through the Amazon Cognito console, you have three options: **Always**, **User Opt-In**, and **No**.

Don't remember

Your user pool doesn't prompt users to remember devices when they sign in.

Always remember

When your app confirms a user's device, your user pool always remembers the device and doesn't return MFA challenges on future successful device sign-ins.

User opt-in

When your app confirms a user's device, your user pool doesn't automatically suppress MFA challenges. You must prompt your user to choose whether they want to remember the device.

When you choose **Always remember** or **User Opt-In**, Amazon Cognito generates a device-identifier key and secret every time a user signs in from an unidentified device. The device key is the initial identifier that your app sends to your user pool when your user performs device authentication.

With each confirmed user device, whether remembered automatically or opted-in, you can use the device-identifier key and secret to authenticate a device on every user sign-in.

You can also configure remembered-device settings for your user pool in a CreateUserPool or UpdateUserPool API request. For more information, see the DeviceConfiguration property.

The Amazon Cognito user pools API has additional operations for remembered devices.

1. ListDevices and AdminListDevices return a list of the device keys and their metadata for a user.
2. GetDevice and AdminGetDevice return the device key and metadata for a single device.
3. UpdateDeviceStatus and AdminUpdateDeviceStatus set a user's device as remembered or not remembered.
4. ForgetDevice and AdminForgetDevice remove a user's confirmed device from their profile.

API operations with names that begin with `Admin` are for use in server-side apps and must be authorized with IAM credentials. For more information, see Understanding API, OIDC, and managed login pages authentication.

Getting a device key

Any time that your user signs in with the user pools API and doesn't include a device key in the authentication parameters as `DEVICE_KEY`, Amazon Cognito returns a new device key in the response. In your public client-side app, place the device key in app storage so that you can include it in future requests. In your confidential server-side app, set a browser cookie or another client-side token with your user's device key.

Before your user can sign in with their trusted device, your app must confirm the device key and provide additional information. Generate a ConfirmDevice request to Amazon Cognito that confirms your user's device with the device key, a friendly name, password verifier, and a salt. If you configured your user pool for opt-in device authentication, Amazon Cognito responds to your `ConfirmDevice` request with a prompt that your user must choose whether to remember the current device. Respond with your user's selection in an UpdateDeviceStatus request.

When you confirm your user's device but don't set it as remembered, Amazon Cognito stores the association but proceeds with non-device sign-in when you provide the device key. Devices

can generate logs that are useful for user security and troubleshooting. A confirmed but unremembered device doesn't take advantage of the sign-in feature, but does take advantage of the security monitoring logs feature. When you activate threat protection for your app client and encode a device fingerprint into your request, Amazon Cognito associates user events with the confirmed device.

To get a new device key

1. Start your user's sign-in session with an InitiateAuth API request.
2. Respond to all authentication challenges with RespondToAuthChallenge until you receive JSON web tokens (JWTs) that mark your user's sign-in session complete.
3. In your app, record the values that Amazon Cognito returns in `NewDeviceMetadata` in its `RespondToAuthChallenge` or `InitiateAuth` response: `DeviceGroupKey` and `DeviceKey`.
4. Generate a new SRP secret for your user: a salt and a password verifier. This function is available in SDKs that provide SRP libraries.
5. Prompt your user for a device name, or generate one from your user's device characteristics.
6. Provide your user's access token, device key, device name, and SRP secret in a ConfirmDevice API request. If your user pool is set to **Always remember** devices, your user's registration is complete.
7. If Amazon Cognito responded to `ConfirmDevice` with `"UserConfirmationNecessary": true`, prompt your user to choose if they would like to remember the device. If they affirm that they want to remember the device, generate an UpdateDeviceStatus API request with your user's access token, device key, and `"DeviceRememberedStatus": "remembered"`.
8. If you have instructed Amazon Cognito to remember the device, the next time they sign in, instead of an MFA challenge, they're presented with a `DEVICE_SRP_AUTH` challenge.

Signing in with a device

After you configure a user's device to be remembered, Amazon Cognito no longer requires them to submit an MFA code when they sign in with the same device key. Device authentication only replaces the MFA-authentication challenge with a device-authentication challenge. You can't sign users in with device authentication only. Your user must first complete authentication with their password or a custom challenge. The following is the authentication process for a user on a remembered device.

To perform device authentication in a flow that uses Custom authentication challenge Lambda triggers, pass a DEVICE_KEY parameter in your InitiateAuth API request. After your user succeeds all challenges and the CUSTOM_CHALLENGE challenge returns an `issueTokens` value of `true`, Amazon Cognito returns one final DEVICE_SRP_AUTH challenge.

To sign in with a device

1. Retrieve your user's device key from client storage.
2. Start your user's sign-in session with an InitiateAuth API request. Choose an `AuthFlow` of USER_SRP_AUTH, REFRESH_TOKEN_AUTH, USER_PASSWORD_AUTH, or CUSTOM_AUTH. In `AuthParameters`, add your user's device key to the DEVICE_KEY parameter, and include the other required parameters for your selected sign-in flow.
   a. You can also pass DEVICE_KEY in the parameters of a PASSWORD_VERIFIER response to an authentication challenge.
3. Complete challenge responses until you receive a DEVICE_SRP_AUTH challenge in the response.
4. In a RespondToAuthChallenge API request, send a `ChallengeName` of DEVICE_SRP_AUTH and parameters for USERNAME, DEVICE_KEY, and SRP_A.
5. Amazon Cognito responds with a DEVICE_PASSWORD_VERIFIER challenge. This challenge response includes values for SECRET_BLOCK and SRP_B.
6. With your SRP library, generate and submit PASSWORD_CLAIM_SIGNATURE, PASSWORD_CLAIM_SECRET_BLOCK, TIMESTAMP, USERNAME, and DEVICE_KEY parameters. Submit these in an additional `RespondToAuthChallenge` request.
7. Complete additional challenges until you receive the user's JWTs.

The following pseudocode demonstrates how to calculate values for your DEVICE_PASSWORD_VERIFIER challenge response. For SRP authentication with a device, generate a *new* SRP secret for your user: a fresh high-entropy password `DeviceSecret`, a salt, and the associated password verifier. These values are distinct from the password, salt, and verifier used for the user's SRP authentication. They are only used for device authentication and are only stored on the device. Functions for generating the SRP secrets for users' devices are available in SRP libraries that are available in various SDKs.

```
PASSWORD_CLAIM_SECRET_BLOCK = SECRET_BLOCK
TIMESTAMP = "Tue May 7 00:09:40 UTC 2025"
k = SHA256(N || g) as a non-negative integer in big-endian
u = SHA256(SRP_A || SRP_B) as a non-negative integer in big-endian
```

```
x = SHA256(salt || SHA256(DeviceGroupKey || DeviceKey || ":" || DeviceSecret)) as a
 non-negative integer in big-endian
S_USER = (SRP_B - k * g^x)^(a + u * x) % N
K_USER = HKDF_HMAC_SHA256(salt=u, ikm=S_USER, info="Caldera Derived Key", length=16
 bytes)
PASSWORD_CLAIM_SIGNATURE = Base64(HMAC_SHA256(key=K_USER, message=(DeviceGroupKey ||
 DeviceKey || PASSWORD_CLAIM_SECRET_BLOCK || TIMESTAMP)))
```

Viewing, updating and forgetting devices

You can implement the following features in your app with the Amazon Cognito API.

1. Display information about a user's current device.
2. Display a list of all of your user's devices.
3. Forget a device.
4. Update a device remembered state.

The access tokens that authorize the API requests in the following descriptions must include the `aws.cognito.signin.user.admin` scope. Amazon Cognito adds a claim for this scope to all access tokens that you generate with the Amazon Cognito user pools API. Third-party IdPs must separately manage devices and MFA for their users who authenticate to Amazon Cognito. In managed login, you can request the `aws.cognito.signin.user.admin` scope, but managed login automatically adds device information to advanced security user logs, and doesn't offer to remember devices.

Display information about a device

You can query information about a user's device to determine if it is still in current use. For example, you might want to deactivate remembered devices after they haven't signed in for 90 days.

- To display your user's device information in a public-client app, submit your user's access key and device key in a GetDevice API request.
- To display your user's device information in a confidential-client app, sign an AdminGetDevice API request with AWS credentials and submit your user's username, device key, and user pool.

Display a list of all your user's devices

You can display a list of all your user's devices and their properties. For example, you might want to verify that the current device matches a remembered device.

- In a public-client app, submit your user's access token in a ListDevices API request.
- In a confidential-client app, sign an AdminListDevices API request with AWS credentials and submit your user's username and user pool.

Forget a device

You can delete a user's device key. You might want to do this when you determine that your user no longer uses a device, or when you detect unusual activity and want to prompt a user to complete MFA again. To register the device again later, you must generate and store a new device key.

- In a public-client app, submit your user's device key and access token in ForgetDevice API request.
- In a confidential-client app, submit your user's device key and access token in AdminForgetDevice API request.

Advanced security with threat protection

After you create your user pool, you have access to **Threat protection** in the navigation menu in the Amazon Cognito console. You can turn threat protection features on and customize the actions that are taken in response to different risks. Or you can use audit mode to gather metrics on detected risks without applying any security mitigations. In audit mode, threat protection publishes metrics to Amazon CloudWatch. You can see metrics after Amazon Cognito generates its first event. See Viewing threat protection metrics.

Threat protection, formerly called *advanced security features*, is a set of monitoring tools for unwanted activity in your user pool, and configuration tools to automatically shut down potentially malicious activity. Threat protection has different configuration options for standard and custom authentication operations. For example, you might want to send a notification to a user with a suspicious custom authentication sign-in, where you have set up additional security factors, but block a user at the same risk level with basic username-password authentication.

Threat protection is available in the Plus feature plan. For more information, see User pool feature plans.

The following user pool options are the components of threat protection.

Compromised credentials

Users reuse passwords for multiple user accounts. The compromised credentials feature of Amazon Cognito compiles data from public leaks of user names and passwords, and compares your users' credentials to lists of leaked credentials. Compromised credentials detection also checks for commonly-guessed passwords. You can check for compromised credentials in username-and-password standard authentication flows in user pools. Amazon Cognito doesn't detect compromised credentials in secure remote password (SRP) or custom authentication.

You can choose the user actions that prompt a check for compromised credentials, and the action that you want Amazon Cognito to take in response. For sign-in, sign-up, and password-change events, Amazon Cognito can **Block sign-in**, or **Allow sign-in**. In both cases, Amazon Cognito generates a user activity log where you can find more information about the event.

Learn more

Working with compromised-credentials detection

Adaptive authentication

Amazon Cognito can review location and device information from your users' sign-in requests and apply an automatic response to secure the user accounts in your user pool against suspicious activity. You can monitor user activity and automate responses to detected risk levels in username-password and SRP, and custom authentication.

When you activate threat protection, Amazon Cognito assigns a risk score to user activity. You can assign an automatic response to suspicious activity: you can **Require MFA**, **Block sign-in**, or just log the activity details and risk score. You can also automatically send email messages that notify your user of the suspicious activity so that they can reset their password or take other self-guided actions.

Learn more

Working with adaptive authentication

IP address allowlist and denylist

With Amazon Cognito threat protection in **Full function** mode, you can create IP address **Always block** and **Always allow** exceptions. A session from an IP address on the **Always block** exception list isn't assigned a risk level by adaptive authentication, and can't sign in to your user pool.

Things to know about IP-address allowlists and blocklists

- You must express **Always block** and **Always allow** in CIDR format, for example `192.0.2.0/24`, a 24-bit mask, or `192.0.2.252/32`, a single IP address.
- Devices with IP addresses in an **Always block** IP range can't sign up or sign in with SDK-based or managed login applications, but they can sign in with third-party IdPs.
- **Always allow** and **Always block** lists don't affect token refresh.
- Amazon Cognito doesn't apply adaptive authentication MFA rules to devices from an **Always allow** IP range, but does apply compromised-credentials rules.

Log export

Threat protection logs granular details of users' authentication requests to your user pool. These logs feature threat assessments, user information, and session metadata like location and device. You can create external archives of these logs for retention and analysis. Amazon Cognito user pools export threat protection logs to Amazon S3, CloudWatch Logs, and Amazon Data Firehose. For more information, see Viewing and exporting user event history.

Learn more

Exporting threat protection user activity logs

Topics

- Considerations and limitations for threat protection
- Turning on threat protection in user pools
- Threat protection enforcement concepts
- Threat protection for standard authentication and custom authentication
- Threat protection prerequisites
- Setting up threat protection
- Working with compromised-credentials detection
- Working with adaptive authentication
- Collecting data for threat protection in applications

Considerations and limitations for threat protection

Threat protection options differ between authentication flows

Amazon Cognito supports both adaptive authentication and compromised-credentials detection with the authentication flows `USER_PASSWORD_AUTH` and `ADMIN_USER_PASSWORD_AUTH`. You can enable only adaptive authentication for `USER_SRP_AUTH`. You can't use threat protection with federated sign-in.

Always-block IPs contribute to request quotas

Blocked requests from IP addresses on an **Always block** exception list in your user pool contribute to the request rate quotas for your user pools.

Threat protection doesn't apply rate limits

Some malicious traffic has the characteristic of a high volume of requests, like distributed denial of service (DDoS) attacks. The risk ratings that Amazon Cognito applies to incoming traffic are per-request and don't take request volume into account. Individual requests in a high-volume event might receive a risk score and an automated response for application-layer reasons that aren't related to their role in a volumetric attack. To implement defenses against volumetric attacks in your user pools, add AWS WAF web ACLs. For more information, see Associate an AWS WAF web ACL with a user pool.

Threat protection doesn't affect M2M requests

Client credentials grants are intended for machine-to-machine (M2M) authorization with no connection to user accounts. Threat protection only monitors user accounts and passwords in your user pool. To implement security features with your M2M activity, consider the capabilities of AWS WAF for monitoring request rates and content. For more information, see Associate an AWS WAF web ACL with a user pool.

Turning on threat protection in user pools

Amazon Cognito user pools console

To activate threat protection for a user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. If you haven't already, activate the Plus feature plan from the **Settings** menu.
5. Choose the **Threat protection** menu and select **Activate**.

6. Choose **Save changes**.

API

Set your feature plan to Plus in a CreateUserPool or UpdateUserPool API request. The following partial example request body sets threat protection to full-function mode. For a complete example request, see Examples.

```
"UserPoolAddOns": {
      "AdvancedSecurityMode": "ENFORCED"
   }
```

Threat protection is the collective term for the features that monitor user operations for signs of account takeover and automatically respond to secure affected user accounts. You can apply threat protection settings to users when they sign in with standard and custom authentication flows.

Threat protection generates logs that detail users' sign-in, sign-out, and other activity. You can export these logs to a third-party system. For more information, see Viewing and exporting user event history.

Threat protection enforcement concepts

Threat protection starts out in an *audit-only* mode where your user pool monitors user activity, assigns risk levels, and generates logs. As a best practice, run in audit-only mode for two weeks or more before you enable *full-function mode*. Full-function mode includes a set of automatic reactions to detected risky activity and compromised passwords. With audit-only mode, you can monitor the threat assessments that Amazon Cognito is performing. You can also provide feedback that trains the feature on false positives and negatives.

You can configure threat protection enforcement at the user pool level to cover all app clients in the user pool, and at the level of individual app clients. App client threat-protection configurations override the user pool configuration. To configure threat protection for an app client, navigate to the app client settings from the **App clients** menu of your user pool in the Amazon Cognito console. There, you can **Use client-level settings** and configure enforcement exclusive to the app client.

Additionally, you can configure threat protection separately for standard and custom authentication types.

Threat protection for standard authentication and custom authentication

The ways that you can configure threat protection depend on the type of authentication you're doing in your user pool and app clients. Each of the following types of authentication can have their own enforcement mode and automated responses.

Standard authentication

Standard authentication is user sign-in, sign-out and password management with username-password flows and in managed login. Amazon Cognito threat protection monitors operations for indicators of risk when they sign in with managed login or use the following API `AuthFlow` parameters:

InitiateAuth

`USER_PASSWORD_AUTH`, `USER_SRP_AUTH`. The compromised credentials feature doesn't have access to passwords in `USER_SRP_AUTH` sign-in, and doesn't monitor or act on events with this flow.

AdminInitiateAuth

`ADMIN_USER_PASSWORD_AUTH`, `USER_SRP_AUTH`. The compromised credentials feature doesn't have access to passwords in `USER_SRP_AUTH` sign-in, and doesn't monitor or act on events with this flow.

You can set the **Enforcement mode** for standard authentication to **Audit only** or **Full function**. To disable threat monitoring for standard authentication, set threat protection to **No enforcement**.

Custom authentication

Custom authentication is user sign-in with custom challenge Lambda triggers. You can't do custom authentication in managed login. Amazon Cognito threat protection monitors operations for indicators of risk when they sign in with the API `AuthFlow` parameter `CUSTOM_AUTH` of `InitiateAuth` and `AdminInitiateAuth`.

You can set the **Enforcement mode** for custom authentication to **Audit only**, **Full function**, or **No enforcement**. The **No enforcement** option disables threat monitoring for custom authentication without affecting other threat protection features.

Threat protection prerequisites

Before you begin, you need the following:

- A user pool with an app client. For more information, see Create a new application in the Amazon Cognito console.
- Set multi-factor authentication (MFA) to **Optional** in the Amazon Cognito console to use the risk-based adaptive authentication feature. For more information, see Adding MFA to a user pool.
- If you're using email notifications, go to the Amazon SES console to configure and verify an email address or domain to use with your email notifications. For more information about Amazon SES, see Verifying Identities in Amazon SES.

Setting up threat protection

Follow these instructions to set up user pool threat protection.

> **Note**
>
> To set up a different threat protection configuration for an app client in the Amazon Cognito user pools console, select the app client from the **App clients** menu and choose **Use client-level settings**.

AWS Management Console

To configure threat protection for a user pool

1. Go to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Threat protection** menu and select **Activate**.
5. Choose the threat protection method that you want to configure: **Standard and custom authentication**. You can set different enforcement modes for custom and standard authentication, but they share the configuration of automated responses in **Full function** mode.
6. Select **Edit**.

7. Choose an **Enforcement mode**. To start responding to detected risks immediately, select **Full function** and configure the automated responses for compromised credentials and adaptive authentication. To gather information in user-level logs and in CloudWatch, select **Audit only** .

   We recommend that you keep threat protection in audit mode for two weeks before enabling actions. During this time, Amazon Cognito can learn the usage patterns of your app users and you can provide event feedback to adjust responses.

8. If you selected **Audit only**, choose **Save changes**. If you selected **Full function**:

   a. Select whether you will take **Custom** action or use or **Cognito defaults** to respond to suspected **Compromised credentials**. **Cognito defaults** are:

      i. Detect compromised credentials on **Sign-in**, **Sign-up**, and **Password change**.

      ii. Respond to compromised credentials with the action **Block sign-in**.

   b. If you selected **Custom** actions for **Compromised credentials**, choose the user pool actions that Amazon Cognito will use for **Event detection** and the **Compromised credentials responses** that you would like Amazon Cognito to take. You can **Block sign-in** or **Allow sign-in** with suspected compromised credentials.

   c. Choose how to respond to malicious sign-in attempts under **Adaptive authentication**. Select whether you will take **Custom** action or use or **Cognito defaults** to respond to suspected malicious activity. When you select **Cognito defaults**, Amazon Cognito blocks sign-in at all risk levels and does not notify the user.

   d. If you selected **Custom** actions for **Adaptive authentication**, choose the **Automatic risk response** actions that Amazon Cognito will take in response to detected risks based on severity level. When you assign a response to a level of risk, you can't assign a less-restrictive response to a higher level of risk. You can assign the following responses to risk levels:

      i. **Allow sign-in** - Take no preventative action.

      ii. **Optional MFA** - If the user has MFA configured, Amazon Cognito will always require the user to provide an additional SMS or time-based one-time password (TOTP) factor when they sign in. If the user does not have MFA configured, they can continue signing in normally.

      iii. **Require MFA** - If the user has MFA configured, Amazon Cognito will always require the user to provide an additional SMS or TOTP factor when they sign in. If the user

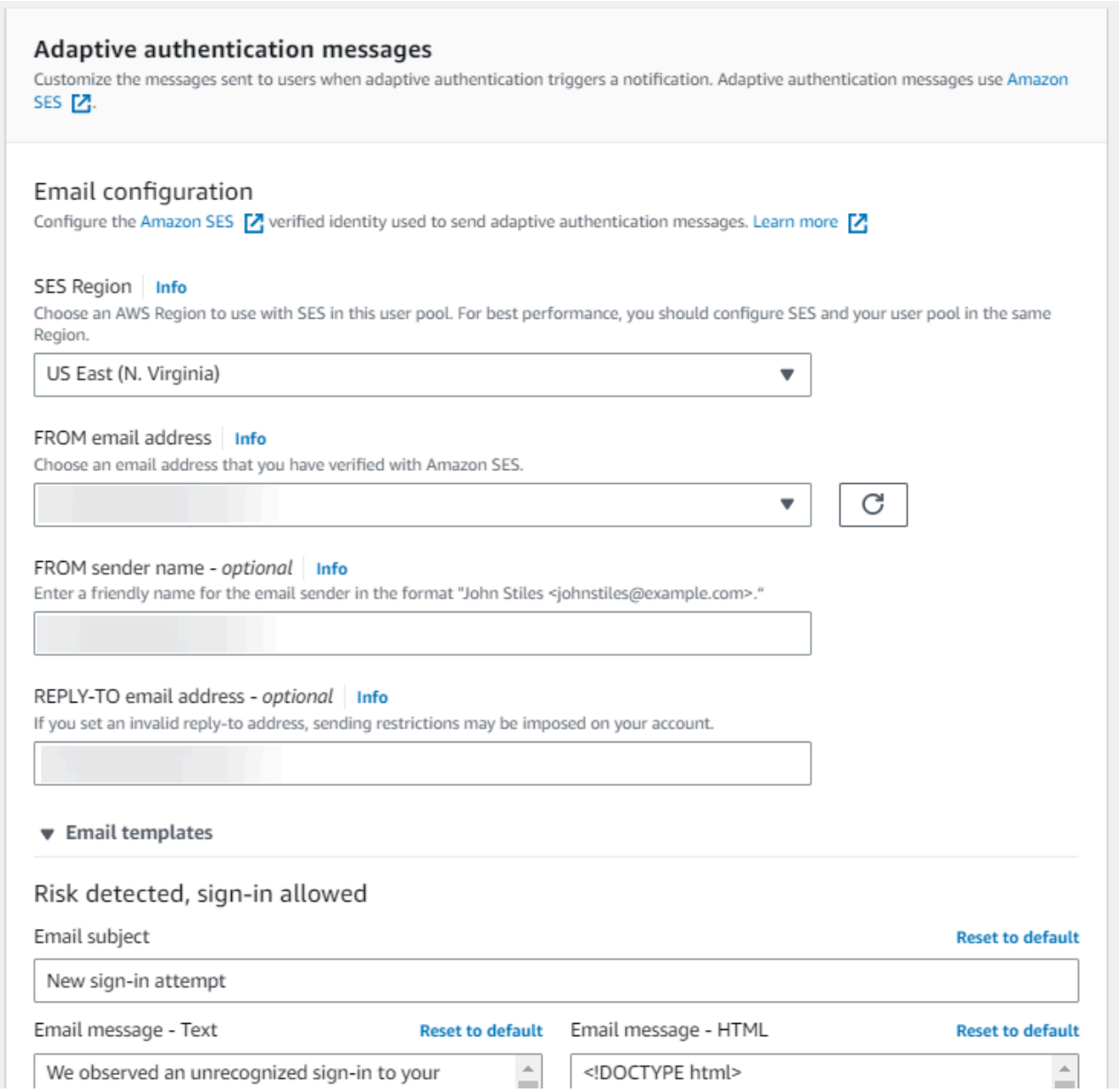
does not have MFA configured, Amazon Cognito will prompt them to set up MFA. Before you automatically require MFA for your users, configure a mechanism in your app to capture phone numbers for SMS MFA, or to register authenticator apps for TOTP MFA.

iv. **Block sign-in** - Prevent the user from signing in.

v. **Notify user** - Send an email message to the user with information about the risk that Amazon Cognito detected and the response you have taken. You can customize email message templates for the messages you send.

9. If you chose **Notify user** in the previous step, you can customize your email delivery settings and email message templates for adaptive authentication.

   a. Under **Email configuration**, choose the **SES Region**, **FROM email address**, **FROM sender name**, and **REPLY-TO email address** that you want to use with adaptive authentication. For more information about integrating your user pool email messages with Amazon Simple Email Service, see Email settings for Amazon Cognito user pools.



b. Expand **Email templates** to customize adaptive authentication notifications with both HTML and plaintext versions of email messages. To learn more about email message templates, see Message templates.

10. Expand **IP address exceptions** to create an **Always-allow** or an **Always-block** list of IPv4 or IPv6 address ranges that will always be allowed or blocked, regardless of the threat protection risk assessment. Specify the IP address ranges in CIDR notation (such as 192.168.100.0/24).
11. Choose **Save changes**.

API (user pool)

To set the threat protection configuration for a user pool, send a SetRiskConfiguration API request that includes a `UserPoolId` parameter, but not a `ClientId` parameter. The following is an example request body for a user pool. This risk configuration takes an escalating series of actions based on the severity of risk and notifies users at all risk levels. It applies a compromised-credentials block to sign-up operations.

To enforce this configuration, you must set `AdvancedSecurityMode` to `ENFORCED` in a separate CreateUserPool or UpdateUserPool API request. For more information about the placeholder templates like `{username}` in this example, see Configuring MFA, authentication, verification and invitation messages.

```
{
   "AccountTakeoverRiskConfiguration": {
      "Actions": {
         "HighAction": {
            "EventAction": "MFA_REQUIRED",
            "Notify": true
         },
         "LowAction": {
            "EventAction": "NO_ACTION",
            "Notify": true
         },
         "MediumAction": {
            "EventAction": "MFA_IF_CONFIGURED",
            "Notify": true
         }
      },
      "NotifyConfiguration": {
         "BlockEmail": {
            "Subject": "You have been blocked for suspicious activity",
            "TextBody": "We blocked {username} at {login-time} from {ip-address}."
         },
         "From": "admin@example.com",
         "MfaEmail": {
            "Subject": "Suspicious activity detected, MFA required",
            "TextBody": "Unexpected sign-in from {username} on device {device-name}.
 You must use MFA."
         },
         "NoActionEmail": {
            "Subject": "Suspicious activity detected, secure your user account",
```

```
            "TextBody": "We noticed suspicious sign-in activity by {username} from
 {city}, {country} at {login-time}. If this was not you, reset your password."
         },
         "ReplyTo": "admin@example.com",
         "SourceArn": "arn:aws:ses:us-west-2:123456789012:identity/
admin@example.com"
      }
   },
   "CompromisedCredentialsRiskConfiguration": {
      "Actions": {
         "EventAction": "BLOCK"
      },
      "EventFilter": [ "SIGN_UP" ]
   },
   "RiskExceptionConfiguration": {
      "BlockedIPRangeList": [ "192.0.2.0/24","198.51.100.0/24" ],
      "SkippedIPRangeList": [ "203.0.113.0/24" ]
   },
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

API (app client)

To set the threat protection configuration for an app client, send a SetRiskConfiguration API request that includes a `UserPoolId` parameter and a `ClientId` parameter. The following is an example request body for an app client. This risk configuration is more severe than the user pool configuration, blocking high-risk entries. It also applies compromised-credentials blocks to sign-up, sign-in, and password-reset operations.

To enforce this configuration, you must set `AdvancedSecurityMode` to `ENFORCED` in a separate CreateUserPool or UpdateUserPool API request. For more information about the placeholder templates like `{username}` in this example, see Configuring MFA, authentication, verification and invitation messages.

```
{
   "AccountTakeoverRiskConfiguration": {
      "Actions": {
         "HighAction": {
            "EventAction": "BLOCK",
            "Notify": true
         },
         "LowAction": {
```

```
            "EventAction": "NO_ACTION",
            "Notify": true
         },
         "MediumAction": {
            "EventAction": "MFA_REQUIRED",
            "Notify": true
         }
      },
      "NotifyConfiguration": {
         "BlockEmail": {
            "Subject": "You have been blocked for suspicious activity",
            "TextBody": "We blocked {username} at {login-time} from {ip-address}."
         },
         "From": "admin@example.com",
         "MfaEmail": {
            "Subject": "Suspicious activity detected, MFA required",
            "TextBody": "Unexpected sign-in from {username} on device {device-name}.
 You must use MFA."
         },
         "NoActionEmail": {
            "Subject": "Suspicious activity detected, secure your user account",
            "TextBody": "We noticed suspicious sign-in activity by {username} from
 {city}, {country} at {login-time}. If this was not you, reset your password."
         },
         "ReplyTo": "admin@example.com",
         "SourceArn": "arn:aws:ses:us-west-2:123456789012:identity/
admin@example.com"
      }
   },
   "ClientId": "1example23456789",
   "CompromisedCredentialsRiskConfiguration": {
      "Actions": {
         "EventAction": "BLOCK"
      },
      "EventFilter": [ "SIGN_UP", "SIGN_IN", "PASSWORD_CHANGE" ]
   },
   "RiskExceptionConfiguration": {
      "BlockedIPRangeList": [ "192.0.2.1/32","192.0.2.2/32" ],
      "SkippedIPRangeList": [ "192.0.2.3/32","192.0.2.4/32" ]
   },
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Working with compromised-credentials detection

Amazon Cognito can detect if a user's username and password have been compromised elsewhere. This can happen when users reuse credentials at more than one site, or when they use insecure passwords. Amazon Cognito checks local users who sign in with username and password, in managed login and with the Amazon Cognito API.

From the **Threat protection** menu of the Amazon Cognito console, you can configure **Compromised credentials**. Configure **Event detection** to choose the user events that you want to monitor for compromised credentials. Configure **Compromised credentials responses** to choose whether to allow or block the user if compromised credentials are detected. Amazon Cognito can check for compromised credentials during sign-in, sign-up, and password changes.

When you choose **Allow sign-in**, you can review Amazon CloudWatch Logs to monitor the evaluations that Amazon Cognito makes on user events. For more information, see Viewing threat protection metrics. When you choose **Block sign-in**, Amazon Cognito prevents sign-in by users who use compromised credentials. When Amazon Cognito blocks sign-in for a user, it sets the user's `UserStatus` to `RESET_REQUIRED`. A user with a `RESET_REQUIRED` status must change their password before they can sign in again.

Compromised credentials can check passwords for the following user activity.

Sign-up

Your user pool checks the passwords that users transmit in the SignUp operation and from the sign-up page of managed login for indicators of compromise.

Sign-in

Your user pool checks passwords that users submit in password-based sign-in for indicators of compromise. Amazon Cognito can review the `ADMIN_USER_PASSWORD_AUTH` flow in AdminInitiateAuth, the `USER_PASSWORD_AUTH` flow in InitiateAuth, and the `PASSWORD` option of the `USER_AUTH` flow in both.

Currently, Amazon Cognito doesn't check for compromised credentials for sign-in operations with Secure Remote Password (SRP) flow. SRP sends a hashed proof of password during sign-in. Amazon Cognito doesn't have access to passwords internally, so it can only evaluate a password that your client passes to it in plaintext.

Password reset

Your user pool checks for indicators of compromise in operations that set new user passwords with the ConfirmForgotPassword self-service password reset operation. The code that's required for this operation is generated by ForgotPassword and AdminResetUserPassword.

Compromised credentials doesn't check temporary or permanent administrator-set passwords set with AdminSetUserPassword. However, with temporary passwords, your user pool checks passwords from responses to the `NEW_PASSWORD_REQUIRED` challenge in RespondToAuthChallenge and AdminRespondToAuthChallenge.

To add compromised credentials protections to your user pool, see Advanced security with threat protection.

Working with adaptive authentication

With adaptive authentication, you can configure your user pool to block suspicious sign-ins or add second factor authentication in response to an increased risk level. For each sign-in attempt, Amazon Cognito generates a risk score for how likely the sign-in request is to be from a compromised source. This risk score is based on device and user factors that your application provides, and others that Amazon Cognito derives from the request. Some factors that contribute to the risk evaluation by Amazon Cognito are IP address, user agent, and geographical distance from other sign-in attempts. Adaptive authentication can turn on or require multi-factor authentication (MFA) for a user in your user pool when Amazon Cognito detects risk in a user's session, and the user hasn't yet chosen an MFA method. When you activate MFA for a user, they always receive a challenge to provide or set up a second factor during authentication, regardless of how you configured adaptive authentication. From your user's perspective, your app offers to help them set up MFA, and optionally Amazon Cognito prevents them from signing in again until they have configured an additional factor.

Amazon Cognito publishes metrics about sign-in attempts, their risk levels, and failed challenges to Amazon CloudWatch. For more information, see Viewing threat protection metrics.

To add adaptive authentication to your user pool, see Advanced security with threat protection.

Topics

- Adaptive authentication overview
- Adding user device and session data to API requests

- Viewing and exporting user event history
- Providing event feedback
- Sending notification messages

Adaptive authentication overview

From the **Threat protection** menu in the Amazon Cognito console, you can choose settings for adaptive authentication, including what actions to take at different risk levels and customization of notification messages to users. You can assign a global threat protection configuration to all of your app clients, but apply a client-level configuration to individual app clients.

Amazon Cognito adaptive authentication assigns one of the following risk levels to each user session: **High**, **Medium**, **Low**, or **No Risk**.

Consider your options carefully when you change your **Enforcement method** from **Audit-only** to **Full-function**. The automatic responses that you apply to risk levels influence the risk level that Amazon Cognito assigns to subsequent user sessions with the same characteristics. For example, after you choose to take no action, or **Allow**, user sessions that Amazon Cognito initially evaluates to be high-risk, Amazon Cognito considers similar sessions to have a lower risk.

For each risk level, you can choose from the following options:

Option	Action
Allow	Users can sign in without an additional factor.
Optional MFA	Users who have a second factor configure d must complete a second factor challenge to sign in. A phone number for SMS and a TOTP software token are the available second factors. Users without a second factor configured can sign in with only one set of credentials.
Require MFA	Users who have a second factor configure d must complete a second factor challenge to sign in. Amazon Cognito blocks sign-in

Option	Action
	for users who don't have a second factor configured.
Block	Amazon Cognito blocks all sign-in attempts at the designated risk level.

> **Note**
>
> You don't have to verify phone numbers to use them for SMS as a second authentication factor.

Adding user device and session data to API requests

You can collect and pass information about your user's session to Amazon Cognito threat protection when you use the API to sign them up, sign them in, and reset their password. This information includes your user's IP address and a unique device identifier.

You might have a intermediate network device between your users and Amazon Cognito, like a proxy service or an application server. You can collect users' context data and pass it to Amazon Cognito so that adaptive authentication calculates your risk based on the characteristics of the user endpoint, instead of your server or proxy. If your client-side app calls Amazon Cognito API operations directly, adaptive authentication automatically records the source IP address. However, it does not record other device information like the `user-agent` unless you also collect a device fingerprint.

Generate this data with the Amazon Cognito context data collection library and submit it to Amazon Cognito threat protection with the ContextData and UserContextData parameters. The context data collection library is included in the AWS SDKs. For more information, see Integrating Amazon Cognito authentication and authorization with web and mobile apps. You can submit `ContextData` if you have the Plus feature plan. For more information, see Setting up threat protection.

When you call the following Amazon Cognito authenticated API operations from your application server, pass the IP of the user's device in the `ContextData` parameter. In addition, pass your server name, server path, and encoded device-fingerprinting data.

- AdminInitiateAuth
- AdminRespondToAuthChallenge

When you call Amazon Cognito unauthenticated API operations, you can submit `UserContextData` to Amazon Cognito threat protection. This data includes a device fingerprint in the `EncodedData` parameter. You can also submit an `IpAddress` parameter in your `UserContextData` if you meet the following conditions:

- Your user pool is on the Plus feature plan. For more information, see User pool feature plans.
- Your app client has a client secret. For more information, see Application-specific settings with app clients.
- You have activated **Accept additional user context data** in your app client. For more information, see Accepting additional user context data (AWS Management Console).

Your app can populate the `UserContextData` parameter with encoded device-fingerprinting data and the IP address of the user's device in the following Amazon Cognito unauthenticated API operations.

- InitiateAuth
- RespondToAuthChallenge
- SignUp
- ConfirmSignUp
- ForgotPassword
- ConfirmForgotPassword
- ResendConfirmationCode

Accepting additional user context data (AWS Management Console)

Your user pool accepts an IP address in a `UserContextData` parameter after you activate the **Accept additional user context data** feature. You don't need to activate this feature if:

- Your users only sign in with authenticated API operations like AdminInitiateAuth , and you use the `ContextData` parameter.
- You only want your unauthenticated API operations to send a device fingerprint, but not an IP address, to Amazon Cognito threat protection.

Update your app client as follows in the Amazon Cognito console to add support for additional user context data.

1. Sign in to the Amazon Cognito console .
2. In the navigation pane, choose **Manage your User Pools**, and choose the user pool you want to edit.
3. Choose the **App clients** menu.
4. Choose or create an app client. For more information, see Configuring a user pool app client.
5. Choose **Edit** from the **App client information** container.
6. In the **Advanced authentication settings** for your app client, choose **Accept additional user context data**.
7. Choose **Save changes**.

To configure your app client to accept user context data in the Amazon Cognito API, set `EnablePropagateAdditionalUserContextData` to `true` in a CreateUserPoolClient or UpdateUserPoolClient request. For information about how to work with threat protection in your web or mobile app, see Collecting data for threat protection in applications. When your app calls Amazon Cognito from your server, collect user context data from the client side. The following is an example that uses the JavaScript SDK method `getData`.

```
var EncodedData =
 AmazonCognitoAdvancedSecurityData.getData(username, userPoolId, clientId);
```

When you design your app to use adaptive authentication, we recommend that you incorporate the latest Amazon Cognito SDK into your app.. The latest version of the SDK collects device fingerprinting information like device ID, model, and time zone. For more information about Amazon Cognito SDKs, see Install a user pool SDK. Amazon Cognito threat protection only saves and assigns a risk score to events that your app submits in the correct format. If Amazon Cognito returns an error response, check that your request includes a valid secret hash and that the `IPaddress` parameter is a valid IPv4 or IPv6 address.

`ContextData` and `UserContextData` resources

- AWS Amplify SDK for Android: GetUserContextData
- AWS Amplify SDK for iOS: userContextData
- JavaScript: amazon-cognito-advanced-security-data.min.js

Viewing and exporting user event history

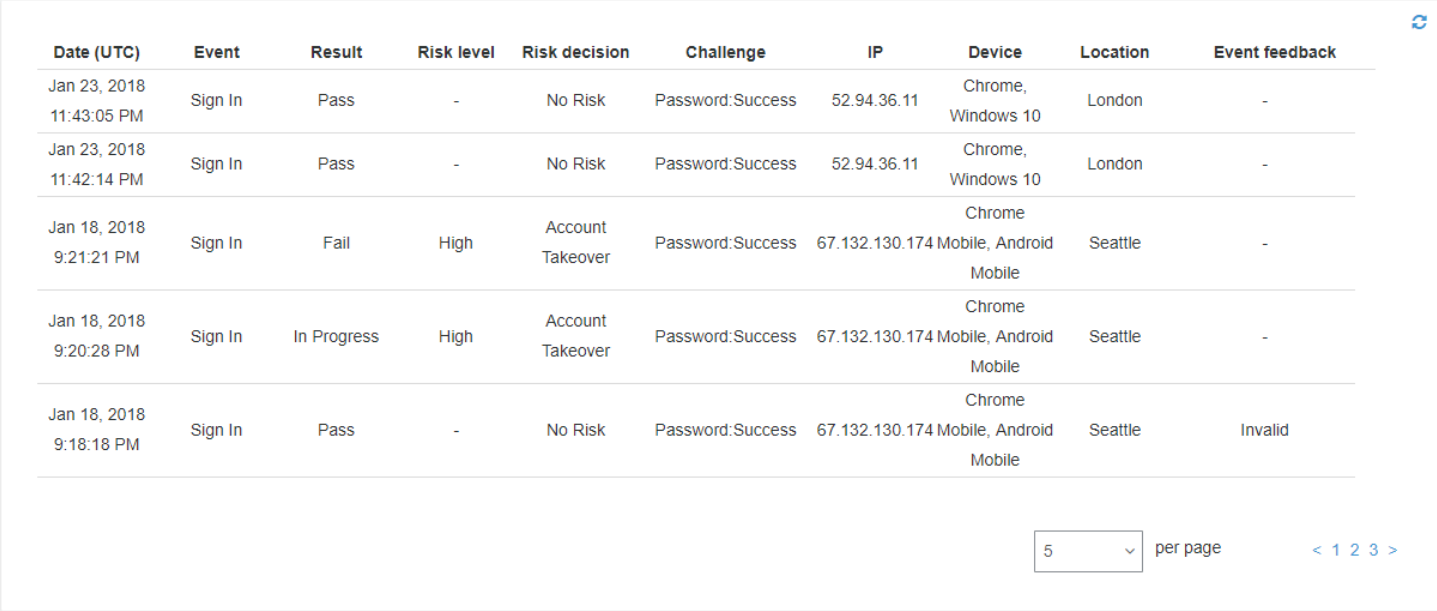
Amazon Cognito generates a log for each authentication event by a user when you enable threat protection. By default, you can view user logs in the **Users** menu in the Amazon Cognito console or with the AdminListUserAuthEvents API operation. You can also export these events to an external system like CloudWatch Logs, Amazon S3, or Amazon Data Firehose. The export feature can make security information about user activity in your application more accessible to your own security-analysis systems.

Topics

- Viewing user event history (AWS Management Console)
- Viewing user event history (API/CLI)
- Exporting user authentication events

Viewing user event history (AWS Management Console)

To see the sign-in history for a user, you can choose the user from the **Users** menu in the Amazon Cognito console. Amazon Cognito retains user event history for two years.



Date (UTC)	Event	Result	Risk level	Risk decision	Challenge	IP	Device	Location	Event feedback
Jan 23, 2018 11:43:05 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10	London	-
Jan 23, 2018 11:42:14 PM	Sign In	Pass	-	No Risk	Password:Success	52.94.36.11	Chrome, Windows 10	London	-
Jan 18, 2018 9:21:21 PM	Sign In	Fail	High	Account Takeover	Password:Success	67.132.130.174	Chrome Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:20:28 PM	Sign In	In Progress	High	Account Takeover	Password:Success	67.132.130.174	Chrome Mobile, Android Mobile	Seattle	-
Jan 18, 2018 9:18:18 PM	Sign In	Pass	-	No Risk	Password:Success	67.132.130.174	Chrome Mobile, Android Mobile	Seattle	Invalid

5 per page < 1 2 3 >

Each sign-in event has an event ID. The event also has corresponding context data, such as location, device details, and risk detection results.

You can also correlate the event ID with the token that Amazon Cognito issued at the time that it recorded the event. The ID and access tokens include this event ID in their payload. Amazon

Cognito also correlates refresh token use to the original event ID. You can trace the original event ID back to the event ID of the sign-in event that resulted in issuing the Amazon Cognito tokens. You can trace token usage within your system to a particular authentication event. For more information, see Understanding user pool JSON web tokens (JWTs).

Viewing user event history (API/CLI)

You can query user event history with the Amazon Cognito API operation AdminListUserAuthEvents or with the AWS Command Line Interface (AWS CLI) with admin-list-user-auth-events.

AdminListUserAuthEvents request

The following request body for `AdminListUserAuthEvents` returns the most recent activity log for one user.

```
{
  "UserPoolId": "us-west-2_EXAMPLE",
  "Username": "myexampleuser",
  "MaxResults": 1
}
```

admin-list-user-auth-events request

The following request for `admin-list-user-auth-events` returns the most recent activity log for one user.

```
aws cognito-idp admin-list-user-auth-events --max-results 1 --username myexampleuser
 --user-pool-id us-west-2_EXAMPLE
```

Response

Amazon Cognito returns the same JSON response body to both requests. The following is an example response for a managed login sign-in event that wasn't found to contain risk factors:

```
{
    "AuthEvents": [
        {
            "EventId": "[event ID]",
            "EventType": "SignIn",
```

```
            "CreationDate": "[Timestamp]",
            "EventResponse": "Pass",
            "EventRisk": {
                "RiskDecision": "NoRisk",
                "CompromisedCredentialsDetected": false
            },
            "ChallengeResponses": [
                {
                    "ChallengeName": "Password",
                    "ChallengeResponse": "Success"
                }
            ],
            "EventContextData": {
                "IpAddress": "192.168.2.1",
                "DeviceName": "Chrome 125, Windows 10",
                "Timezone": "-07:00",
                "City": "Bellevue",
                "Country": "United States"
            }
        }
    ],
    "NextToken": "[event ID]#[Timestamp]"
}
```

Exporting user authentication events

Configure your user pool to export user events from threat protection to an external system. The supported external systems–Amazon S3, CloudWatch Logs, and Amazon Data Firehose–might add costs to your AWS bill for data that you send or retrieve. For more information, see Exporting threat protection user activity logs.

AWS Management Console

1. Sign in to the Amazon Cognito console.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Log streaming** menu. Select **Edit**.
5. Under **Logging status**, select the checkbox next to **Activate user activity log export**.
6. Under **Logging destination**, choose the AWS service that you want to handle your logs: **CloudWatch log group**, **Amazon Data Firehose stream**, or **S3 bucket**.

7. Your selection will populate the resource selector with the corresponding resource type. Select a log group, stream, or bucket from the list. You can also select the **Create** button to navigate to the AWS Management Console for the selected service and create a new resource.
8. Select **Save changes**.

API

Choose one type of destination for your user activity logs.

The following is an example `SetLogDeliveryConfiguration` request body that sets a Firehose stream as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "FirehoseConfiguration": {
            "StreamArn": "arn:aws:firehose:us-west-2:123456789012:deliverystream/
example-user-pool-activity-exported"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

The following is an example `SetLogDeliveryConfiguration` request body that sets a Amazon S3 bucket as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "S3Configuration": {
            "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
```

```
}
```

The following is an example `SetLogDeliveryConfiguration` request body that sets a CloudWatch log group as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "CloudWatchLogsConfiguration": {
            "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:DOC-
EXAMPLE-LOG-GROUP"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

Providing event feedback

Event feedback affects risk evaluation in real time and improves the risk evaluation algorithm over time. You can provide feedback on the validity of sign-in attempts through the Amazon Cognito console and API operations.

> **Note**
>
> Your event feedback influences the risk level that Amazon Cognito assigns to subsequent user sessions with the same characteristics.

In the Amazon Cognito console, choose a user from the **Users** menu and select **Provide event feedback**. You can review the event details and **Set as valid** or **Set as invalid**.

The console lists the sign-in history in user details in the **Users** menu. If you select an entry, you can mark the event as valid or not valid. You can also provide feedback through the user pool API operation AdminUpdateAuthEventFeedback, and through the AWS CLI command admin-update-auth-event-feedback.

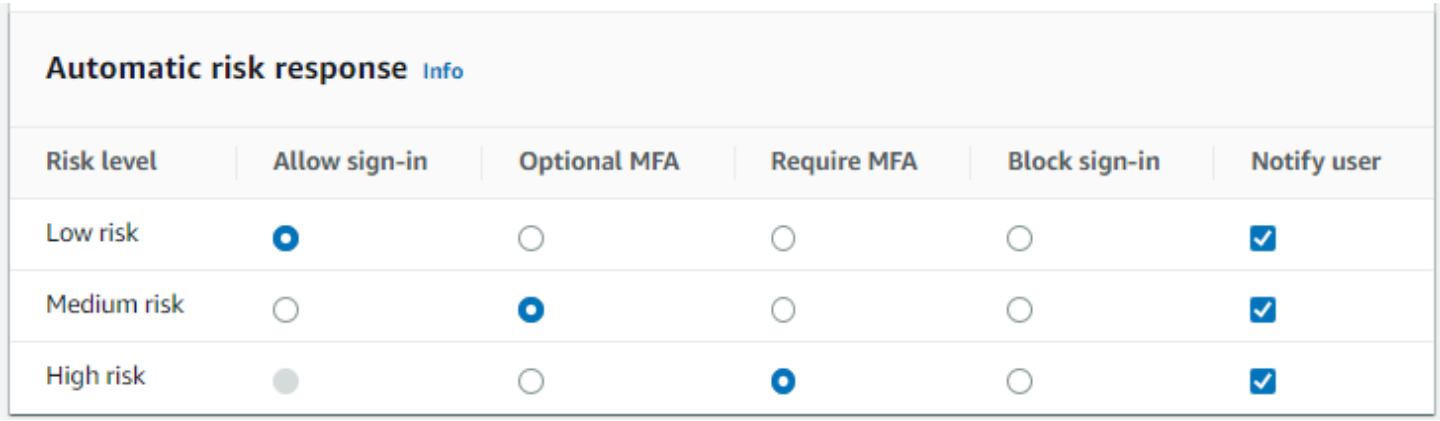
When you select **Set as valid** in the Amazon Cognito console or provide a `FeedbackValue` value of `valid` in the API, you tell Amazon Cognito that you trust a user session where Amazon Cognito has evaluated some level of risk. When you select **Set as invalid** in the Amazon Cognito console or provide a `FeedbackValue` value of `invalid` in the API, you tell Amazon Cognito that you don't trust a user session, or you don't believe that Amazon Cognito evaluated a high-enough risk level.

Sending notification messages

With threat protection, Amazon Cognito can notify your users of risky sign-in attempts. Amazon Cognito can also prompt users to select links to indicate if the sign-in was valid or not valid. Amazon Cognito uses this feedback to improve the risk detection accuracy for your user pool.

> **Note**
>
> Amazon Cognito only sends notification messages to users when their action generates an automated risk response: block sign-in, allow sign-in, set MFA to optional, or require MFA. Some requests might be assigned a level of risk but don't generate adaptive authentication automated risk responses; for these, your user pool doesn't send notifications. For example, incorrect passwords might be logged with a risk rating, but the response by Amazon Cognito is to fail sign-in, not to apply an adaptive authentication rule.

In the **Automatic risk response** section choose **Notify Users** for low, medium, or high-risk cases.



Amazon Cognito sends email notifications to your users regardless of whether they have verified their email address.

You can customize notification email messages, and provide both plaintext and HTML versions of these messages. To customize your email notifications, open **Email templates** from **Adaptive**

authentication messages in your threat protection configuration. To learn more about email templates, see Message templates.

Collecting data for threat protection in applications

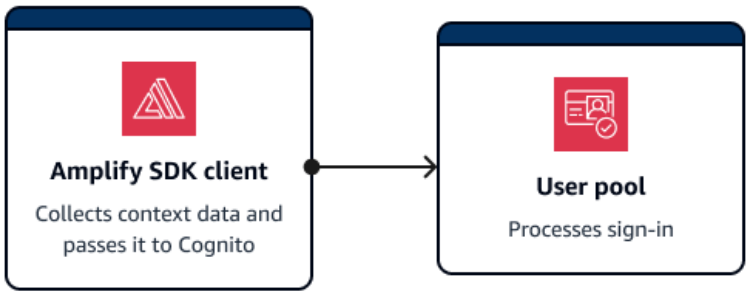
Amazon Cognito adaptive authentication evaluates risk levels for attempted account takeover from contextual details of users' sign-in attempts. Your application must add *context data* to API requests so that Amazon Cognito threat protection can more accurately evaluate risk. Context data is information like IP address, browser agent, device information, and request headers that provides contextual information about how a user connected to your user pool.

The central responsibility of an application that submits this context to Amazon Cognito is an `EncodedData` parameter in authentication requests to user pools. To add this data to your requests, you can implement Amazon Cognito with an SDK that automatically generates this information for you, or you can implement a module for JavaScript, iOS, or Android that collects this data. *Client-only* applications that make direct requests to Amazon Cognito must implement AWS Amplify SDKs. *Client-server* applications that have an intermediate server or API component must implement a separate SDK module.

In the following scenarios, your authentication front end manages user context data collection without any additional configuration:

- Managed login automatically collects and submits context data to threat protection.
- All AWS Amplify libraries have context-data collection built into their authentication methods.

Submitting user context data in client-only applications with Amplify



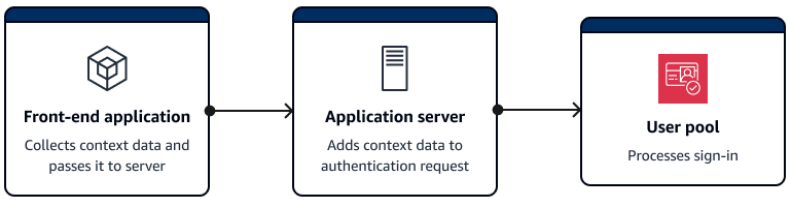
Amplify SDKs support mobile clients that authenticate with Amazon Cognito directly. Clients of this kind make direct API requests to Amazon Cognito public API operations. Amplify clients automatically collect context data for threat protection by default.

Amplify applications with JavaScript are an exception. They require the addition of a JavaScript module that collects user context data.

Typically, an application in this configuration uses unauthenticated API operations like InitiateAuth and RespondToAuthChallenge. The UserContextData object helps evaluate risks more accurately for these operations. The Amplify SDKs add device and session information to an `EncodedData`parameter of `UserContextData`.

Collecting context data in client-server applications

Some applications have a front-end tier that collects user authentication data and an application back-end tier that submits authentication requests to Amazon Cognito. This is a common architecture in webservers and applications backed by microservices. In these applications, you must import a public context-data collection library.



Typically, an application server in this configuration uses authenticated API operations like AdminInitiateAuth and AdminRespondToAuthChallenge. The ContextData object helps Amazon Cognito evaluate risks more accurately for these operations. . The contents of `ContextData` are the encoded data that your front end passed to your server, and additional details from the user's HTTP request to your server. These additional context details, like the HTTP headers and IP address, provide your application server with the characteristics of the user's environment.

Your application server might also do sign-in with unauthenticated API operations like InitiateAuth and RespondToAuthChallenge. The UserContextData object informs threat protection risk analysis in these operations. The operations in the available public context data collection libraries add security information to the `EncodedData` parameter in authentication requests. Additionally, configure your user pool to accept additional context data and add the user's source IP to the `IpAddress` parameter of `UserContextData`.

To add context data to client-server applications

1. In your front-end application, collect encoded context data from the client with an iOS, Android, or JavaScript module.
2. Pass the encoded data and the details of the authentication request to your application server.
3. In your application server, extract the user's IP address, relevant HTTP headers, requested server name, and requested path from the HTTP request. Populate these values to the ContextData parameter of your API request to Amazon Cognito.
4. Populate the `EncodedData` parameter of `ContextData` in your API request with the encoded device data that your SDK module collected. Add this context data to the authentication request.

Context data libraries for client-server applications

JavaScript

The `amazon-cognito-advanced-security-data.min.js` module collects `EncodedData` that you can pass to your application server.

Add the `amazon-cognito-advanced-security-data.min.js` module to your JavaScript configuration. Replace `<region>` with an AWS Region from the following list: `us-east-1`, `us-east-2`, `us-west-2`, `eu-west-1`, `eu-west-2`, or `eu-central-1`.

```
<script src="https://amazon-cognito-assets.<region>.amazoncognito.com/amazon-cognito-
advanced-security-data.min.js"></script>
```

To generate an `encodedContextData` object that you can use in the `EncodedData` parameter, add the following to your JavaScript application source:

```
var encodedContextData = AmazonCognitoAdvancedSecurityData.getData(_username,
 _userpoolId, _userPoolClientId);
```

iOS/Swift

To generate context data, iOS applications can integrate the Mobile SDK for iOS module AWSCognitoIdentityProviderASF.

To collect encoded context data for threat protection, add the following snippet to your application:

```
import AWSCognitoIdentityProviderASF

let deviceId = getDeviceId()
let encodedContextData = AWSCognitoIdentityProviderASF.userContextData(
                            userPoolId,
                            username: username,
                            deviceId: deviceId,
                            userPoolClientId: userPoolClientId)

/**
 * Reuse DeviceId from keychain or generate one for the first time.
 */
func getDeviceId() -> String {
    let deviceIdKey = getKeyChainKey(namespace: userPoolId, key:
 "AWSCognitoAuthAsfDeviceId")

   if let existingDeviceId = self.keychain.string(forKey: deviceIdKey) {
        return existingDeviceId
    }

    let newDeviceId = UUID().uuidString
    self.keychain.setString(newDeviceId, forKey: deviceIdKey)
    return newDeviceId
}

/**
 * Get a namespaced keychain key given a namespace and key
 */
func getKeyChainKey(namespace: String, key: String) -> String {
    return "\(namespace).\(key)"
}
```

Android

To generate context data, Android applications can integrate the Mobile SDK for Android module aws-android-sdk-cognitoidentityprovider-asf.

To collect encoded context data for threat protection, add the following snippet to your application:

```
UserContextDataProvider provider = UserContextDataProvider.getInstance();
// context here is android application context.
```

```
String encodedContextData = provider.getEncodedContextData(context, username,
 userPoolId, userPoolClientId);
```

Associate an AWS WAF web ACL with a user pool

AWS WAF is a web application firewall. With an AWS WAF web access control list (web ACL), you can protect your user pool from unwanted requests to your classic hosted UI, managed login, and Amazon Cognito API service endpoints. A web ACL gives you fine-grained control over all of the HTTPS web requests that your user pool responds to. For more information about AWS WAF web ACLs, see Managing and using a web access control list (web ACL) in the *AWS WAF Developer Guide*.

When you have an AWS WAF web ACL associated with a user pool, Amazon Cognito forwards selected non-confidential headers and contents of requests from your users to AWS WAF. AWS WAF inspects the contents of the request, compares it to the rules that you specified in your web ACL, and returns a response to Amazon Cognito. The request components that Amazon Cognito forwards to AWS WAF depend on the type of request. For managed login and the classic hosted UI, Amazon Cognito doesn't forward the request body. For more information, see Things to know about AWS WAF web ACLs and Amazon Cognito.

Things to know about AWS WAF web ACLs and Amazon Cognito

- You can't configure web ACL rules to match on personally identifiable information (PII) in user pool requests, for example usernames, passwords, phone numbers, or email addresses. This data won't be available to AWS WAF. Instead, configure your web ACL rules to match on session data in the headers, path, and body like IP addresses, browser agents, and requested API operations.
- Amazon Cognito supports body-based web ACL rules only for requests to the user pools API. For managed login and the classic hosted UI, Amazon Cognito doesn't forward the request body and inspects only the headers and other non-body components.
- Web ACL rules can return your custom block response only when the browser loads a managed login page. Requests that the page sends without reloading, such as when a user submits the sign-in form, receive a default `403` block response instead.
- Requests blocked by AWS WAF do not count towards the request rate quota for any request type. The AWS WAF handler is called before the API-level throttling handlers.
- When you create a web ACL, a small amount of time passes before the web ACL has fully propagated and is available to Amazon Cognito. The propagation time can be from a few seconds to a number of minutes. AWS WAF returns a `WAFUnavailableEntityException` when you attempt to associate a web ACL before it has fully propagated.

- You can associate one web ACL with each user pool.
- Your request might result in a payload that is larger than the limits of what AWS WAF can inspect. See Oversize request component handling in the *AWS WAF Developer Guide* to learn how to configure how AWS WAF handles oversize requests from Amazon Cognito.
- You can't associate a web ACL that uses AWS WAF Fraud Control account takeover prevention (ATP) with an Amazon Cognito user pool. The ATP feature is in the `AWS-AWSManagedRulesATPRuleSet` managed rule group. Before you associate a web ACL with a user pool, be sure that it doesn't use this managed rule group.
- When you have an AWS WAF web ACL associated with a user pool, and a rule in your web ACL presents a CAPTCHA, this can cause an unrecoverable error in managed login TOTP registration. To create a rule that has a CAPTCHA action and doesn't affect managed login TOTP, see Configuring your AWS WAF web ACL for managed login TOTP MFA.
- When a rule in your web ACL returns a custom block response, Amazon Cognito adds a Content-Security-Policy header to that response. This policy allows scripts only from the same origin as your user pool domain and from the AWS WAF CAPTCHA and challenge scripts that Amazon Cognito provides. It blocks any other inline or third-party script, including a script that your custom response body uses to redirect the user. For more information, see Content-Security-Policy on custom block responses.
- When a rule in your web ACL returns a custom block response with an HTTP status code of 500 or higher in response to a managed login or classic hosted UI request, Amazon Cognito returns a `403 (Forbidden)` status code instead. For more information, see Status codes in custom block responses.

AWS WAF inspects requests to the following endpoints.

Managed login and the classic hosted UI

Requests to all endpoints in the User pool endpoints and managed login reference.

Public API operations

Requests from your app to the Amazon Cognito API that don't use AWS credentials to authorize. This includes API operations like InitiateAuth, RespondToAuthChallenge, and GetUser. The API operations that are in scope of AWS WAF don't require authentication with AWS credentials. They are unauthenticated, or authorized with a session string or access token. For more information, see List of API operations grouped by authorization model.

You can configure the rules in your web ACL with rule actions that **Count**, **Allow**, **Block**, or present a **CAPTCHA** in response to a request that matches a rule. For more information, see AWS WAF rules in the *AWS WAF Developer Guide*. Depending on the rule action, you can customize the response that Amazon Cognito returns to your users.

> **Important**
>
> Your options to customize the error response depends on the way you make an API request.
>
> - You can customize the error code and response body of managed login requests. Amazon Cognito returns custom error codes of 500 and higher as `403`. For more information, see Status codes in custom block responses. You can only present a CAPTCHA for your user to solve in managed login.
> - For requests that you make with the Amazon Cognito user pools API, you can customize the response body of a request that receives a **Block** response. You can also specify a custom error code in the range 400–499.
> - The AWS Command Line Interface (AWS CLI) and the AWS SDKs return a `ForbiddenException` error to requests that produce a **Block** or **CAPTCHA** response.

Content-Security-Policy on custom block responses

If your custom block response relies on scripts, this policy can affect how it behaves. When a web ACL rule returns a custom block response to managed login or the classic hosted UI, Amazon Cognito adds a Content-Security-Policy (CSP) header to that response. CSP is a browser security mechanism that restricts which scripts and other resources a page can load. The policy restricts the response as follows:

- The response can include scripts and other resources from the same origin as your user pool domain.
- The response can include the AWS WAF CAPTCHA and challenge scripts that Amazon Cognito provides. Amazon Cognito allowlists these scripts so that AWS WAF CAPTCHA and challenge rule actions continue to work.

Any other inline script, and any script hosted on a domain other than your user pool domain, doesn't run. This includes a script that a custom response body uses to redirect the user, submit a form, or send a network request.

If your custom response needs to redirect the user, return an HTTP 3xx status code and a `Location` header instead of a script-based redirect. This pattern doesn't depend on script execution and isn't affected by the CSP.

This behavior applies in all AWS Regions where you can associate a web ACL with a user pool.

Status codes in custom block responses

Amazon Cognito doesn't return `5xx` status codes from AWS WAF custom block responses. When a web ACL rule returns a custom block response with a status code of 500 or higher, Amazon Cognito responds with a `403 (Forbidden)` status code instead. The response still includes the headers and body from your custom response. This behavior applies as follows:

- **Managed login pages** – When the browser loads a managed login page, Amazon Cognito returns `403` with the headers and body from your custom response. Requests that the page sends without reloading, such as form submissions, receive the default `403` block response without your custom headers or body.
- **Classic hosted UI pages and other user pool endpoints** – Amazon Cognito returns `403` with the headers and body from your custom response for classic hosted UI pages. The identity provider and relying party endpoints behave the same way, whether your user pool uses managed login or classic hosted UI branding.
- **User pools API** – The user pools API returns your custom status code only when it's in the range 400–499, and returns `403` for other status codes.

Associating a web ACL with your user pool

To work with a web ACL in your user pool, your AWS Identity and Access Management (IAM) principal must have the following Amazon Cognito and AWS WAF permissions. For information about AWS WAF permissions, see AWS WAF API permissions in the *AWS WAF Developer Guide*.

JSON

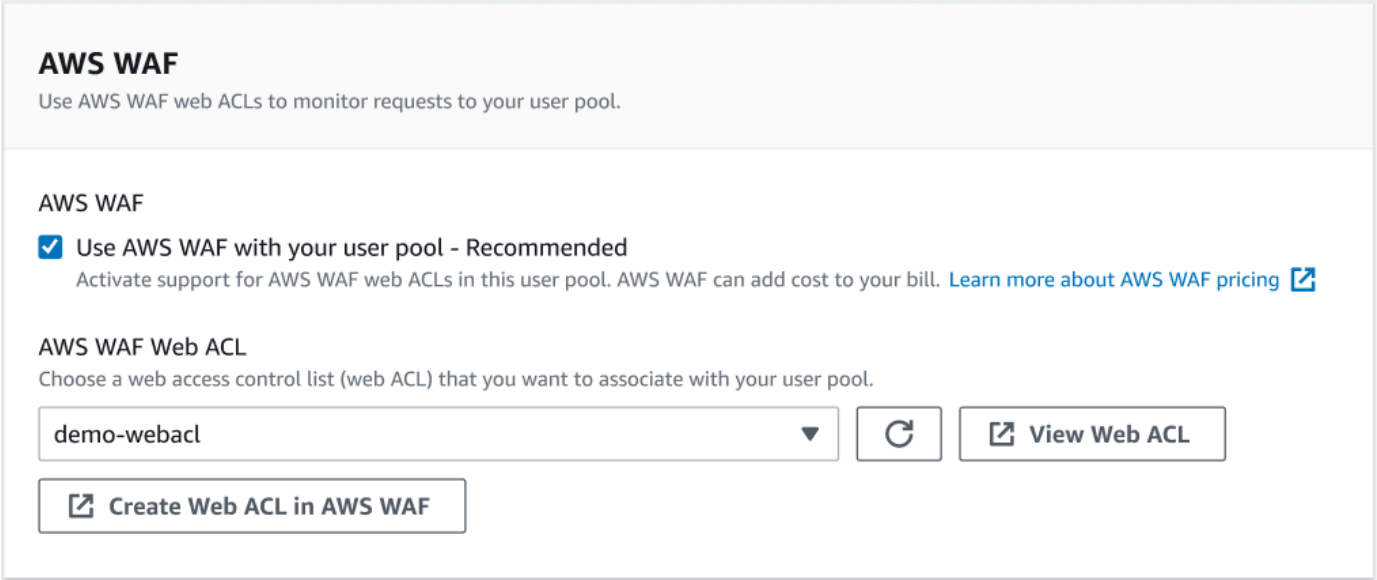
```
{
 "Version":"2012-10-17",
 "Statement": [
  {
   "Sid": "AllowWebACLUserPool",
   "Effect": "Allow",
```

```
    "Action": [
     "cognito-idp:ListResourcesForWebACL",
     "cognito-idp:GetWebACLForResource",
     "cognito-idp:AssociateWebACL"
    ],
    "Resource": [
     "arn:aws-eusc:cognito-idp:*:123456789012:userpool/*"
    ]
   },
   {
    "Sid": "AllowWebACLUserPoolWAFv2",
    "Effect": "Allow",
    "Action": [
     "wafv2:ListResourcesForWebACL",
     "wafv2:AssociateWebACL",
     "wafv2:DisassociateWebACL",
     "wafv2:GetWebACLForResource"
    ],
    "Resource": "arn:aws-eusc:wafv2:*:123456789012:*/webacl/*/*"
   },
   {
    "Sid": "DisassociateWebACL1",
    "Effect": "Allow",
    "Action": "wafv2:DisassociateWebACL",
    "Resource": "*"
   },
   {
    "Sid": "DisassociateWebACL2",
    "Effect": "Allow",
    "Action": [
     "cognito-idp:DisassociateWebACL"
    ],
    "Resource": [
     "arn:aws-eusc:cognito-idp:*:123456789012:userpool/*"
    ]
   }
  ]
}
```

Though you must grant IAM permissions, the listed actions are permission-only and don't correspond to any API operation.

To activate AWS WAF for your user pool and associate a web ACL

1. Sign in to the Amazon Cognito console .
2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Choose the **AWS WAF** tab in the **Security** section.
4. Choose **Edit**.
5. Select **Use AWS WAF with your user pool**.



6. Choose an **AWS WAF Web ACL** that you already created, or choose **Create web ACL in AWS WAF** to create one in a new AWS WAF session in the AWS Management Console.
7. Choose **Save changes**.

To programmatically associate a web ACL with your user pool in the AWS Command Line Interface or an SDK, use AssociateWebACL from the AWS WAF API. Amazon Cognito doesn't have a separate API operation that associates a web ACL.

Testing and logging AWS WAF web ACLs

When you set a rule action to **Count** in your web ACL, AWS WAF adds the request to a count of requests that match the rule. To test a web ACL with your user pool, set rule actions to **Count** and consider the volume of requests that match each rule. For example, if a rule that you want to set to a **Block** action matches a large number of requests that you determine to be normal user traffic, you might need to reconfigure your rule. For more information, see Testing and tuning your AWS WAF protections in the *AWS WAF Developer Guide.*

You can also configure AWS WAF to log request headers to an Amazon CloudWatch Logs log group, an Amazon Simple Storage Service (Amazon S3) bucket, or an Amazon Data Firehose. You can identify the Amazon Cognito requests that you make with the user pools API by the `x-amzn-cognito-client-id` and `x-amzn-cognito-operation-name`. Managed login requests only include the `x-amzn-cognito-client-id` header. For more information, see Logging web ACL traffic in the *AWS WAF Developer Guide*.

AWS WAF web ACLs are available in all user pool feature plans. The security features of AWS WAF complement Amazon Cognito threat protection. You can activate both features in a user pool. AWS WAF bills separately for the inspection of user pool requests. For more information, see AWS WAF Pricing.

Logging AWS WAF request data is subject to additional billing by the service where you target your logs. For more information, see Pricing for logging web ACL traffic information in the *AWS WAF Developer Guide*.

User pool case sensitivity

Amazon Cognito user pools that you create in the AWS Management Console are case insensitive by default. When a user pool is case insensitive, *user@example.com* and *User@example.com* refer to the same user. When usernames in a user pool are case insensitive, the `preferred_username` and `email` attributes also are case insensitive.

Case insensitivity applies not only to attribute inputs, but outputs too. Mixed-case attribute values in case-insensitive user pools are flattened to lowercase in user pool text output. Examples of user pool text output are userInfo responses, user query responses like the output of GetUser, and input events to Lambda triggers.

To account for user pool case sensitivity settings, identify users in your app code based on an alternative user attribute. Because the case of a username, preferred username, or email address attribute can vary in different user profiles, refer instead to the `sub` attribute. You can also create an immutable custom attribute in your user pool, and assign your own unique identifier value to the attribute in each new user profile. When you first create a user, you can write a value to the immutable custom attribute that you created.

> **Note**
>
> Regardless of the case sensitivity settings of your user pool, Amazon Cognito requires that a federated user from a SAML or OIDC identity provider (IdP) pass a unique and

case-sensitive `NameId` or `sub` claim. For more information about unique identifier case sensitivity and SAML IdPs, see Implement SP-initated SAML sign-in.

Creating a case-sensitive user pool

If you create resources with the AWS Command Line Interface (AWS CLI) and API operations such as CreateUserPool, you must set the Boolean `CaseSensitive` parameter to `false`. This setting creates a case-insensitive user pool. If you do not specify a value, `CaseSensitive` defaults to `true`. User pools that you create in the Amazon Cognito console are not case-sensitive. To produce a case-sensitive user pool, you must use the `CreateUserPool` operation. Before February 12, 2020, user pools defaulted to case sensitive regardless of platform.

In the **Sign-in** menu of the AWS Management Console and in the `UsernameConfiguration` property of DescribeUserPool, you can review the case sensitivity settings for each user pool in your account.

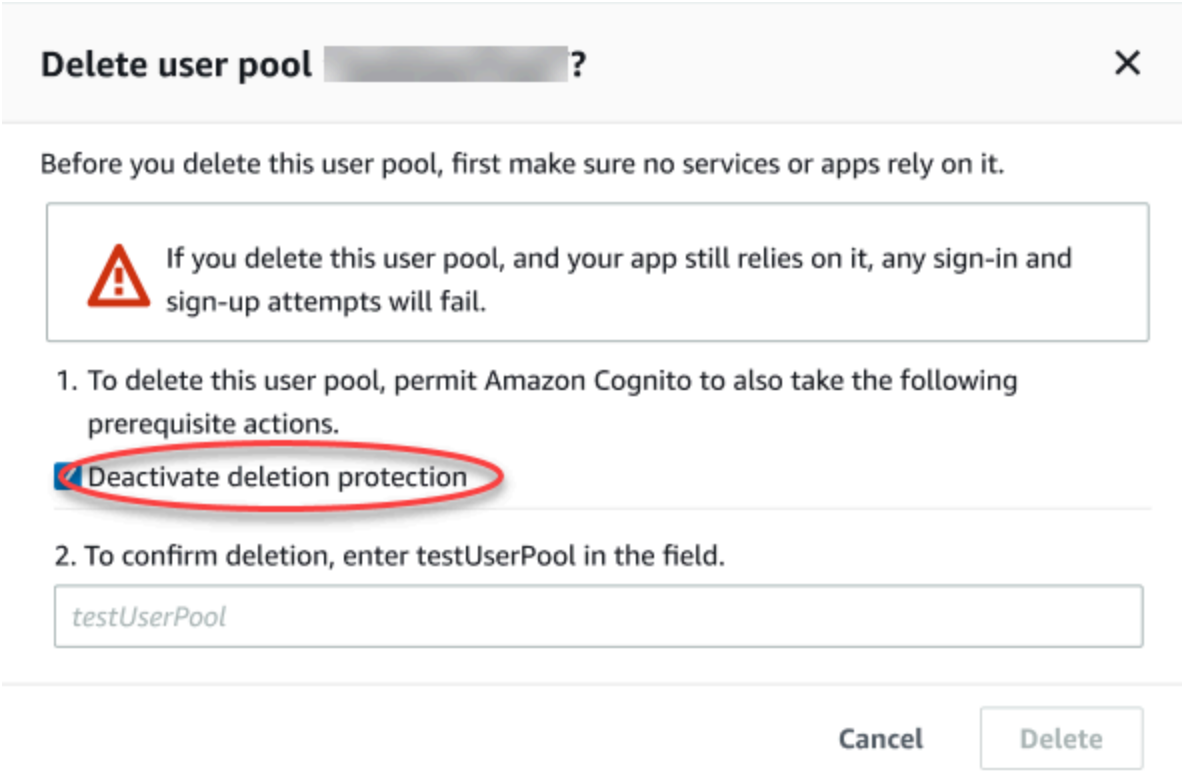
Migrating to a new user pool

Because of potential conflicts between user profiles, you can't change an Amazon Cognito user pool from case-sensitive to case-insensitive. Instead, migrate your users to a new user pool. You must build migration code to resolve case-related conflicts. This code must either return a unique new user or reject the sign-in attempt when it detects a conflict. In a new case-insensitive user pool, assign a Migrate user Lambda trigger. The AWS Lambda function can create users in the new case-insensitive user pool. When the user fails sign-in with the case-insensitive user pool, the Lambda function finds and duplicates the user from the case-sensitive user pool. You can also activate a migrate user Lambda trigger on ForgotPassword events. Amazon Cognito passes user information and event metadata from the sign-in or password-recovery action to your Lambda function. You can use event data to manage conflicts between usernames and email addresses when your function creates the new user in your case-insensitive user pool. These conflicts are between usernames and email addresses that would be unique in a case-sensitive user pool, but identical in a case-insensitive user pool.

For more information about how to use a migrate user Lambda trigger between Amazon Cognito user pools, see Migrating Users to Amazon Cognito user pools in the AWS blog.

User pool deletion protection

To make it so that your administrators don't accidentally delete your user pool, activate deletion protection. With deletion protection active, you can only delete your user pool after you confirm that you want to delete it. When you delete a user pool in the AWS Management Console, you can deactivate deletion protection at the same time. When you accept the prompt to deactivate deletion protection and confirm your intention to delete, as shown in the following image, Amazon Cognito deletes your user pool.



When you want to delete a user pool with an Amazon Cognito API request, you must first change `DeletionProtection` to `Inactive` in an UpdateUserPool request. If you don't deactivate deletion protection, Amazon Cognito returns an `InvalidParameterException` error. After you deactivate deletion protection, you can delete the user pool in a DeleteUserPool request.

Amazon Cognito activates **Deletion protection** by default when you create a new user pool in the AWS Management Console. When you create a user pool with the `CreateUserPool` API, deletion protection is inactive by default. To use this feature in user pools that you create with the AWS CLI or an AWS SDK, set the `DeletionProtection` parameter to `True`.

You can activate or deactivate deletion protection status in the **Deletion protection** container in the **Settings** menu in the Amazon Cognito console.

To configure deletion protection

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Settings** menu, navigate to the **User pool security** tab, and locate **Deletion Protection**. Select **Activate** or **Deactivate**.
5. Confirm your choice in the next dialogue.

Managing user existence error responses

Amazon Cognito supports customizing error responses returned by user pools. Custom error responses are available for user creation and authentication, password recovery, and confirmation operations.

Use the `PreventUserExistenceErrors` setting of a user pool app client to enable or disable user existence related errors. When you create a new app client with the Amazon Cognito user pools API, `PreventUserExistenceErrors` is `LEGACY`, or disabled, by default. In the Amazon Cognito console, the option **Prevent user existence errors** —a setting of `ENABLED` for `PreventUserExistenceErrors`—is selected by default. To update your `PreventUserExistenceErrors` configuration, do one of the following:

- Change the value of `PreventUserExistenceErrors` between `ENABLED` and `LEGACY` in an UpdateUserPoolClient API request.
- Edit your app client in the Amazon Cognito console and change the state of **Prevent user existence errors** between selected (`ENABLED`) and deselected (`LEGACY`).

When this property has a value of `LEGACY`, your app client returns a `UserNotFoundException` error response when a user attempts to sign in with a username that doesn't exist in your user pool.

When this property has a value of `ENABLED`, your app client doesn't disclose the nonexistence of a user account in your user pool with a `UserNotFoundException` error. A `PreventUserExistenceErrors` configuration of `ENABLED` has the following effects when you submit a request for a username that doesn't exist:

- Amazon Cognito responds with nonspecific information to API requests where its response might otherwise disclose that a valid user exists.

- Amazon Cognito returns a generic authentication failure response to forgot-password requests, and to authentication requests with authentication flows *except* for choice-based authentication (USER_AUTH)—for example, USER_SRP_AUTH or CUSTOM_AUTH. The error response tells you the user name or password is incorrect.
- Amazon Cognito responds to requests for choice-based authentication with a random selection from the challenge types allowed for the user pool. Your user pool might return a passkey, one-time password, or password challenge.
- The behavior of Amazon Cognito account confirmation and password recovery APIs alternates between returning a response indicating a code was sent to a simulated delivery medium and returning an `InvalidParameterException` error.

The following information details the behaviors of user pool operations when `PreventUserExistenceErrors` is set to ENABLED.

Authentication and user creation operations

You can configure error responses in username-password, and Secure Remote Password (SRP) authentication. You can also customize the errors that you return with custom authentication. Choice-based authentication is unaffected by your `PreventUserExistenceErrors` configuration.

User-existence disclosure details in authentication flows

Choice-based authentication

In the USER_AUTH choice-based authentication flow, Amazon Cognito returns a challenge from the primary authentication factors that are available, depending on your user pool configuration and users' attributes. This authentication flow can return password, secure remote password (SRP), WebAuthn (passkey), SMS one-time password (OTP), or email OTP challenges. With `PreventUserExistenceErrors` active, Amazon Cognito issues a challenge to nonexistent users to complete one or more of the available forms of authentication. With `PreventUserExistenceErrors` inactive, Amazon Cognito returns a `UserNotFound` exception.

Username and password authentication

The authentication flows ADMIN_USER_PASSWORD_AUTH, USER_PASSWORD_AUTH, and the PASSWORD flow of USER_AUTH return a `NotAuthorizedException` with the

message `Incorrect username or password` when `PreventUserExistenceErrors` is active. When `PreventUserExistenceErrors` is inactive, these flows return `UserNotFoundException`.

Secure Remote Password (SRP) based authentication

As a best practice, only implement `PreventUserExistenceErrors` with `USER_SRP_AUTH` or the `PASSWORD_SRP` flow of `USER_AUTH` in user pools without email address, phone number, or preferred username alias attributes. Users with alias attributes might not be subject to user-existence suppression in the SRP authentication flow. Username-password authentication flows—`ADMIN_USER_PASSWORD_AUTH`, `USER_PASSWORD_AUTH`, and the `USER_AUTH` `PASSWORD` challenge—fully suppress the existence of users from alias attributes.

When someone attempts SRP sign-in with a username that isn't known to your app client, Amazon Cognito returns a simulated response in the first step as described in RFC 5054. Amazon Cognito returns the same salt and an internal user ID in UUID format for the same username and user pool combination. When you send a `RespondToAuthChallenge` API request with proof of password, Amazon Cognito returns a generic `NotAuthorizedException` error when either username or password is incorrect. For more information about implementation of SRP authentication, see Sign-in with persistent passwords and secure payload.

> **Note**
>
> You can simulate a generic response with username and password authentication if you are using verification-based alias attributes, and the immutable username isn't formatted as a UUID.

Custom authentication challenge Lambda trigger

Amazon Cognito invokes the custom authentication challenge Lambda triggers when users attempt to sign in with the `CUSTOM_AUTH` authentication flow, but their username isn't found. The input event includes a boolean parameter named `UserNotFound` with a value of `true` for any nonexistent user. This parameter appears in the request events that your user pool sends to the create, define, and verify auth challenge Lambda functions that make up the custom-authentication architecture. When you examine this indicator in the logic of your Lambda function, you can simulate custom authentication challenges for a user that doesn't exist.

Pre authentication Lambda trigger

Amazon Cognito invokes the pre authentication trigger when users attempt to sign in but their username isn't found. The input event includes a `UserNotFound` parameter with a value of `true` for any nonexistent user.

The following list describes the effect of `PreventUserExistenceErrors` on user account creation.

User-existence disclosure details in user creation flows

SignUp

The `SignUp` operation always returns `UsernameExistsException` when a username is already taken. If you don't want Amazon Cognito to return a `UsernameExistsException` error for email addresses and phone numbers when you sign up users in your app, use verification-based alias attributes. For more information about aliases, see Customizing sign-in attributes.

For an example of how Amazon Cognito can prevent the use of `SignUp` API requests to discover users in your user pool, see Preventing `UsernameExistsException` errors for email addresses and phone numbers on sign-up.

Imported users

If `PreventUserExistenceErrors` is enabled, during authentication of imported users a generic `NotAuthorizedException` error is returned indicating either the username or password was incorrect instead of returning `PasswordResetRequiredException`. See Requiring imported users to reset their passwords for more information.

Migrate user Lambda trigger

Amazon Cognito returns a simulated response for users that don't exist when an empty response was set in the original event context by the Lambda trigger. For more information, see Importing users with a user migration Lambda trigger.

Preventing `UsernameExistsException` errors for email addresses and phone numbers on sign-up

The following example demonstrates how, when you configure alias attributes in your user pool, you can keep duplicate email addresses and phone numbers from generating

`UsernameExistsException` errors in response to `SignUp` API requests. You must have created your user pool with email address or phone number as an alias attribute. For more information, see the *Customizing sign-in attributes* section of User pool attributes.

1. Jie signs up for a new username, and also provides the email address `jie@example.com`. Amazon Cognito sends a code to their email address.

   Example AWS CLI command

   ```
   aws cognito-idp sign-up --client-id 1234567890abcdef0 --username jie --password
    PASSWORD --user-attributes Name="email",Value="jie@example.com"
   ```

   Example response

   ```
   {
       "UserConfirmed": false,
       "UserSub": "<subId>",
       "CodeDeliveryDetails": {
           "AttributeName": "email",
           "Destination": "j****@e****",
           "DeliveryMedium": "EMAIL"
       }
   }
   ```

2. Jie provides the code sent to them to confirm their ownership of the email address. This completes their registration as a user.

   Example AWS CLI command

   ```
   aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=jie --
   confirmation-code xxxxxx
   ```

3. Shirley registers a new user account and provides the email address `jie@example.com`. Amazon Cognito doesn't return a `UsernameExistsException` error, and sends a confirmation code to Jie's email address.

   Example AWS CLI command

   ```
   aws cognito-idp sign-up --client-id 1234567890abcdef0 --username shirley --password
    PASSWORD --user-attributes Name="email",Value="jie@example.com"
   ```

Example response

```
{
    "UserConfirmed": false,
    "UserSub": "<new subId>",
    "CodeDeliveryDetails": {
        "AttributeName": "email",
        "Destination": "j****@e****",
        "DeliveryMedium": "EMAIL"
    }
}
```

4. In a different scenario, Shirley has ownership of `jie@example.com`. Shirley retrieves the code that Amazon Cognito sent to Jie's email address and attempts to confirm the account.

Example AWS CLI command

```
aws cognito-idp confirm-sign-up --client-id 1234567890abcdef0 --username=shirley --
confirmation-code xxxxxx
```

Example response

```
An error occurred (AliasExistsException) when calling the ConfirmSignUp operation: An
 account with the email already exists.
```

Amazon Cognito doesn't return an error to Shirley's `aws cognito-idp sign-up` request, despite `jie@example.com` being assigned to an existing user. Shirley must demonstrate ownership of the email address before Amazon Cognito returns an error response. In a user pool with alias attributes, this behavior prevents use of the public `SignUp` API to check whether a user exists with a given email address or phone number.

This behavior is different from the response that Amazon Cognito returns to `SignUp` request with an existing username, as shown in the following example. While Shirley learns from this response that a user already exists with the username `jie`, they don't learn about any email addresses or phone numbers associated with the user.

Example CLI command

```
aws cognito-idp sign-up --client-id 1example23456789 --username jie --password PASSWORD
```

```
      --user-attributes Name="email",Value="shirley@example.com"
```

Example response

```
An error occurred (UsernameExistsException) when calling the SignUp operation: User
 already exists
```

Password reset operations

Amazon Cognito returns the following responses to user password reset operations when you prevent user existence errors.

ForgotPassword

When a user isn't found, is deactivated, or doesn't have a verified delivery mechanism to recover their password, Amazon Cognito returns `CodeDeliveryDetails` with a simulated delivery medium for a user. The simulated delivery medium is determined by the input username format and verification settings of the user pool.

ConfirmForgotPassword

Amazon Cognito returns the `CodeMismatchException` error for users that don't exist or are disabled. If a code isn't requested when using `ForgotPassword`, Amazon Cognito returns the `ExpiredCodeException` error.

Confirmation operations

Amazon Cognito returns the following responses to user confirmation and verification operations when you prevent user existence errors.

ResendConfirmationCode

Amazon Cognito returns `CodeDeliveryDetails` for a disabled user or a user that doesn't exist. Amazon Cognito sends a confirmation code to the existing user's email or phone number.

ConfirmSignUp

`ExpiredCodeException` returns if a code has expired. Amazon Cognito returns `NotAuthorizedException` when a user isn't authorized. If the code doesn't match what the server expects Amazon Cognito returns `CodeMismatchException`.

Building B2B multi-tenant applications

This section describes methods that you can implement to separate tenants between Amazon Cognito resources within the same Region and AWS account. You can also split your tenants across more than one AWS account or Region, and give each of them their own quota. Other advantages of multi-Region multi-tenancy include the highest possible level of isolation, shortest network-transit time for globally distributed users, and adherence to existing distribution models in your organization.

Amazon Cognito user pools operate with multi-tenant applications that generate a volume of requests that must remain within Amazon Cognito quotas. To scale up this capacity when your customer base grows, you can purchase additional quota capacity.

> **Note**
>
> Amazon Cognito quotas are applied per AWS account and AWS Region. These quotas are shared across all tenants in your application. Review the Amazon Cognito service quotas, and make sure that the quota meets the expected volume and the expected number of tenants in your application.

Single-Region multi-tenancy can also have advantages for your customers and administrators.

The following list covers some of the advantages of multi-tenancy with shared resources.

Advantages of multi-tenancy

Common user directory

Multi-tenancy supports models where customers have accounts in more than one application. You can link identities from third-party providers into a single consistent user pool profile. In cases where user profiles are unique to their tenant, any multi-tenancy strategy with a single user pool has one point of entry to user administration.

Common security

In a shared user pool, you can create a single standard for security and apply the same threat protection, multi-factor authentication (MFA), and AWS WAF standards to all tenants. Because an AWS WAF web ACL must be in the same AWS Region as the resource that you associate it with, multi-tenancy offers shared access to a complex resource. When you want to maintain

consistent security configuration in multi-Region Amazon Cognito applications, you must apply operational standards that replicate your configuration between resources.

Common customization

You can customize user pools and identity pools with AWS Lambda. Configuration of Lambda triggers in user pools and Amazon Cognito events in identity pools can become complex. Lambda functions must be in the same AWS Region as your user pool or identity pool. Shared Lambda functions can enforce standards for custom authentication flows, user migration, token generation, and other functions within a Region.

Common messaging

Amazon Simple Notification Service (Amazon SNS) requires additional configuration in a Region before you can send SMS messages to your users. You can send email messages with Amazon Simple Email Service (Amazon SES) verified identities and domains that are contained within a Region.

With multi-tenancy, you can share this configuration and maintenance overhead between all of your tenants. Because Amazon SNS and Amazon SES aren't available in all AWS Regions, splitting your resources between Regions requires additional consideration.

When you use custom messaging providers, you gain the *common customization* of a single Lambda function to manage your message delivery.

Managed login sets a session cookie in the browser so that it recognizes a user who has already authenticated. When you authenticate *local users* in a user pool, their session cookie authenticates them for all app clients in the same user pool. A local user exists exclusively in your user pool directory without federation through an external IdP. The session cookie is valid for one hour. You can't change the session cookie duration.

There are two ways to prevent sign-in across app clients with a hosted UI session cookie.

- Separate your users into per-tenant user pools.
- Replace hosted UI sign-in with Amazon Cognito user pools API sign-in.

Topics

- User-pool multi-tenancy best practices
- App-client multi-tenancy best practices

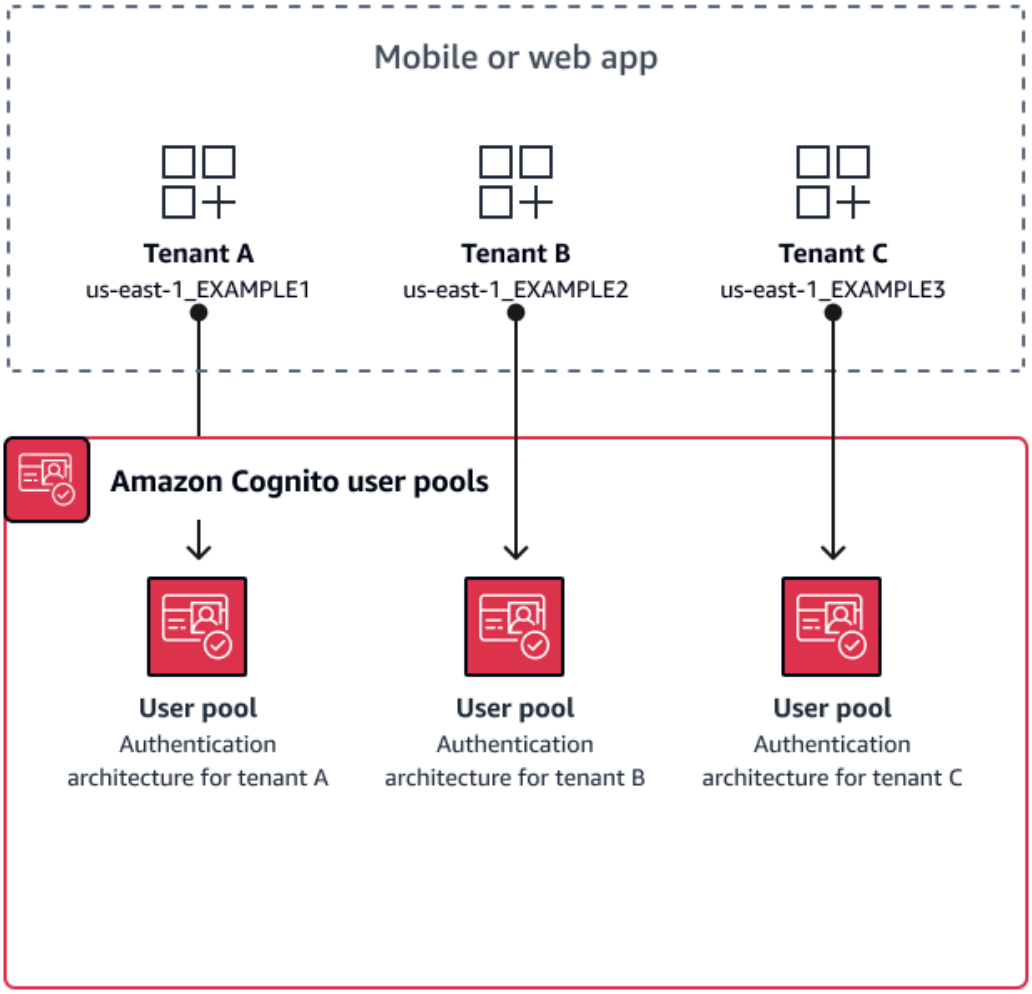
- User group multi-tenancy best practices
- Custom-attribute multi-tenancy best practices
- Custom scope multi-tenancy best practices
- Multi-tenancy security recommendations

User-pool multi-tenancy best practices

Create a user pool for each tenant in your app. This approach provides maximum isolation for each tenant. You can implement different configurations for each tenant. Tenant isolation by user pool gives you flexibility in user-to-tenant mapping. You can create multiple profiles for the same user. However, each user must sign up individually for each tenant they can access.

Using this approach, you can set up a hosted UI for each tenant independently and redirect users to their tenant-specific instance of your application. You can also use this approach to integrate with backend services like Amazon API Gateway.

The following diagram shows each tenant with a dedicated user pool.



When to implement user-pool multi-tenancy

When isolation and customization are your primary concerns. The relationship between users and tenants might be complex in an architecture with multiple user pools. Consider an example where you have two educational tenants. The same user might be a limited-access student in one app, and a teacher with a high level of permissions in another. You might require MFA in one app but not another, or have a different password policy. Because local users can sign in to multiple app clients in user pools with managed login, user-pool multi-tenancy is also ideal when you want more than one of your tenants to sign in with managed login.

Level of effort

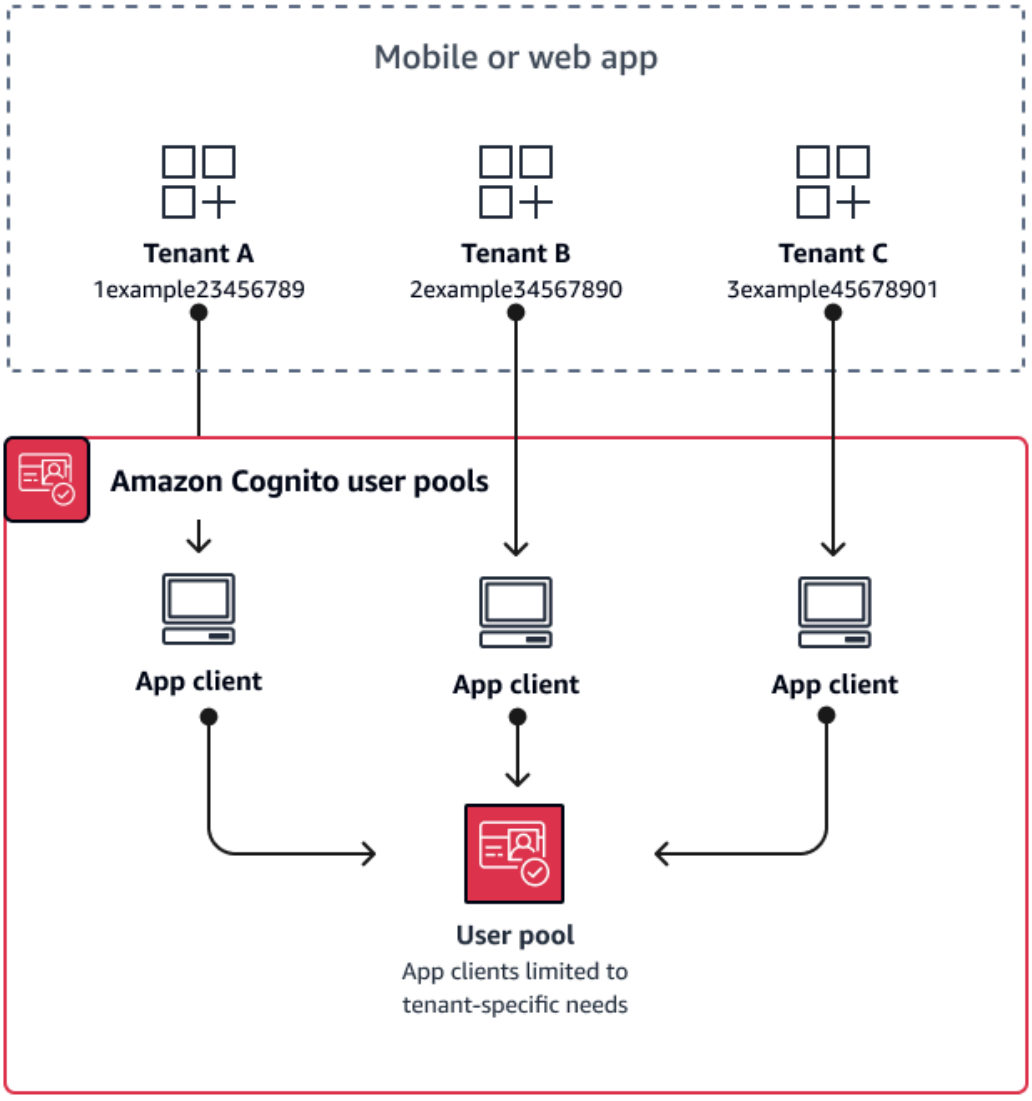
The development and operation effort to use this approach is high. To ensure consistent and predictable outcomes for your family of apps, you must integrate Amazon Cognito resources with

your automation tools and maintain your baselines as your authentication architecture grows more complex. When you want to create a single starting place for your apps, you have to build the user-interface (UI) elements to capture the initial decision that routes users to the correct resource.

App-client multi-tenancy best practices

Create an app client for each tenant in your app. With app-client multi-tenancy, you can assign any user to tenant-linked app clients and retain a single user profile. Because you can assign any or all of the identity providers (IdPs) in your user pool to an app client, a tenant app client can permit sign-in with a tenant-specific IdP. When users exist in multiple tenants, you can link their profiles with multiple IdPs for a consistent user experience.

The following diagram shows each tenant with a dedicated app client in a shared user pool.



When to implement app-client multi-tenancy

When you can choose a universal configuration for settings at the user-pool level, like Lambda triggers, password policy, and the content and delivery methods of email and SMS messages. Because users in a shared user pool can sign in to any app client, app-client multi-tenancy is ideal for sign-in with app-client-specific IdPs or the Amazon Cognito user pools API. App-client multi-tenancy is also well-suited for one-to-many environments where you want to permit users to transition between multiple applications.

Level of effort

App-client multi-tenancy requires moderate effort. A major challenge of app-client multi-tenancy is the ability for tenants to present a hosted UI cookie and switch between apps. In an app-client multi-tenancy architecture, avoid hosted UI sign-in where isolation is necessary. You can distribute your mobile app or links to your web app with app client logic built in, or you can build initial UI elements that determine users' tenancy. The level of effort is lower because you don't need to standardize and maintain configuration across multiple user pools and identity pools.

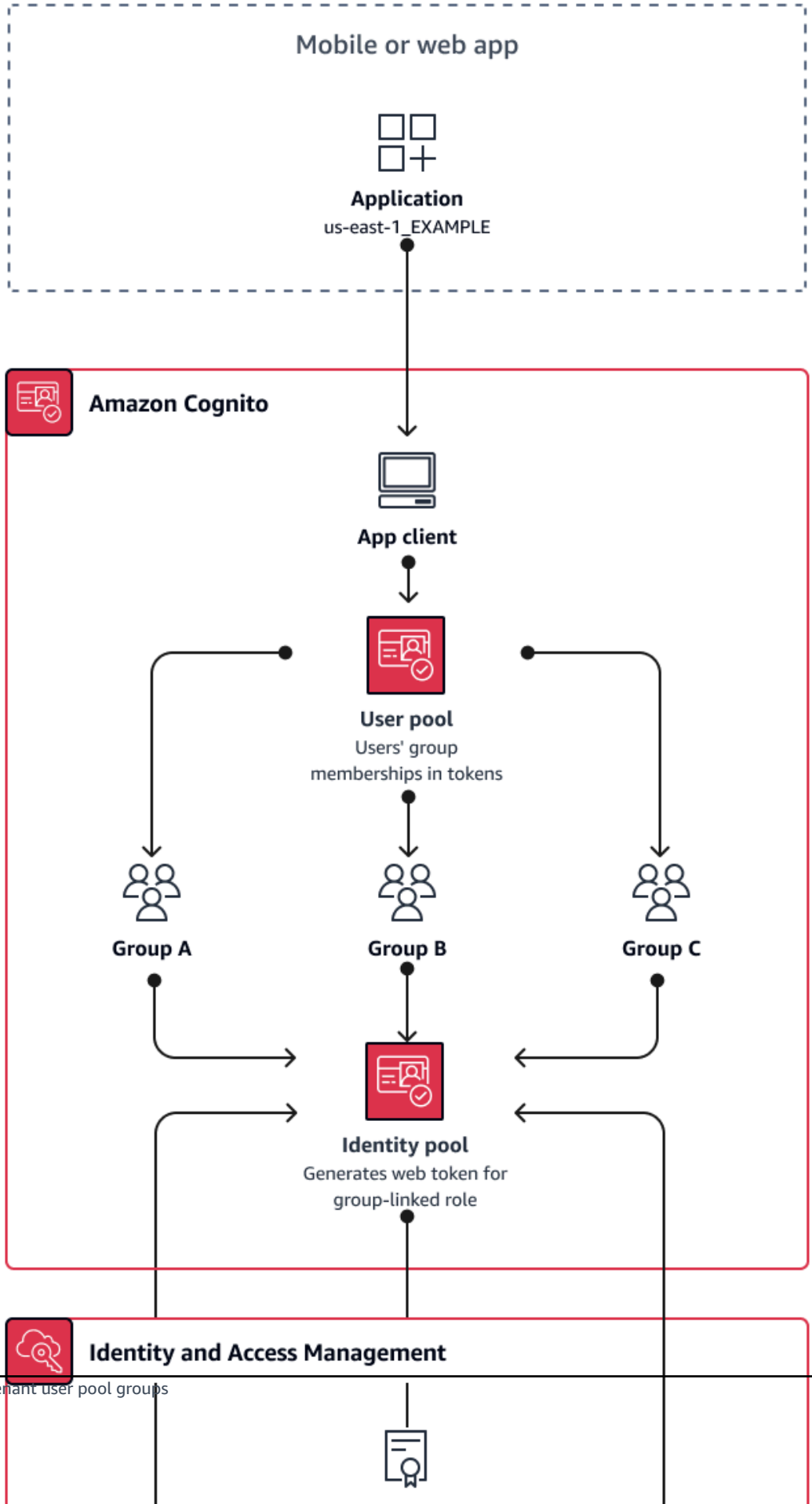
User group multi-tenancy best practices

Group-based multi-tenancy works best when your architecture requires Amazon Cognito user pools with identity pools.

User pool ID and access tokens contain a `cognito:groups` claim. Additionally, ID tokens contain `cognito:roles` and `cognito:preferred_role` claims. When the primary outcome of authentication in your app is temporary AWS credentials from an identity pool, your users' group memberships can determine the IAM role and permissions that they receive.

As an example, consider three tenants that each store application assets in their own Amazon S3 bucket. Assign the users of each tenant to an associated group, configure a preferred role for the group, and grant that role read access to their bucket.

The following diagram shows tenants sharing an app client and a user pool, with dedicated groups in the user pool that determine their eligibility for an IAM role.


Mobile or web app
Application
us-east-1_EXAMPLE
Amazon Cognito
App client
User pool
Users' group memberships in tokens
Group A
Group B
Group C
Identity pool
Generates web token for group-linked role
Identity and Access Management

When to implement group multi-tenancy

When access to AWS resources is your primary concern. Groups in Amazon Cognito user pools are a mechanism for role-based access control (RBAC). You can configure many groups in a user pool and make complex RBAC decisions with group priority. Identity pools can assign credentials for the role with the highest priority, any role in the groups claim, or from other claims in a user's tokens.

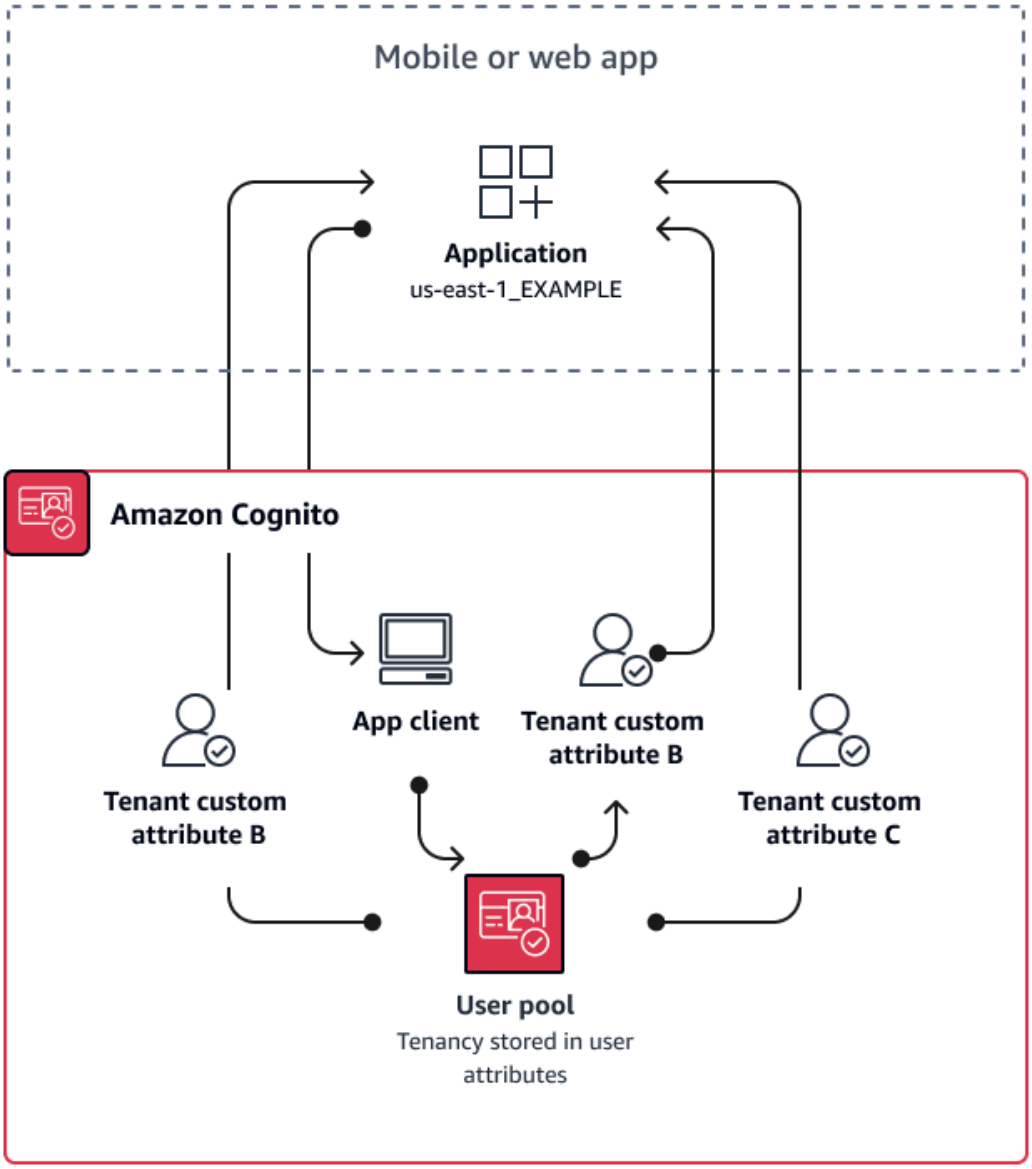
Level of effort

The level of effort to maintain multi-tenancy with group membership alone is low. However, to expand the role of user pool groups beyond the built-in capacity for IAM role selection, you must build application logic that processes group membership in users' tokens, and determine what to do in the client. You can integrate Amazon Verified Permissions with your apps to make client-side authorization decisions. Group identifiers aren't currently processed in Verified Permissions IsAuthorizedWithToken API operations, but you can develop custom code that parses the contents of tokens, including group-membership claims.

Custom-attribute multi-tenancy best practices

Amazon Cognito supports custom attributes with names that you choose. One scenario where custom attributes are useful is when they distinguish the tenancy of users in a shared user pool. When you assign users a value for an attribute like `custom:tenantID`, your app can assign access to tenant-specific resources accordingly. A custom attribute that defines a tenant ID should be immutable or read-only to the app client.

The following diagram shows tenants sharing an app client and a user pool, with a custom attributes in the user pool that indicates the tenant that they belong to.



When custom attributes determine tenancy, you can distribute a single application or sign-in URL. After your user signs in, your app can process the `custom:tenantID` claim determine which assets to load, the branding to apply, and features to display. For advanced access-control decisions from user attributes, set up your user pool as an identity provider in Amazon Verified Permissions, and generate access decisions from the contents of ID or access tokens.

When to implement custom-attribute multi-tenancy

When tenancy is surface-level. A tenant attribute can contribute to branding and layout outcomes. When you want to achieve significant isolation between tenants, custom attributes aren't the best choice. Any difference between tenants that must be configured at the user-pool or app-client level, like MFA or hosted UI branding, requires that you create distinctions between tenants in a way that custom attributes don't offer. With identity pools, you can even choose the IAM role from your users from the custom-attribute claim in their ID token.

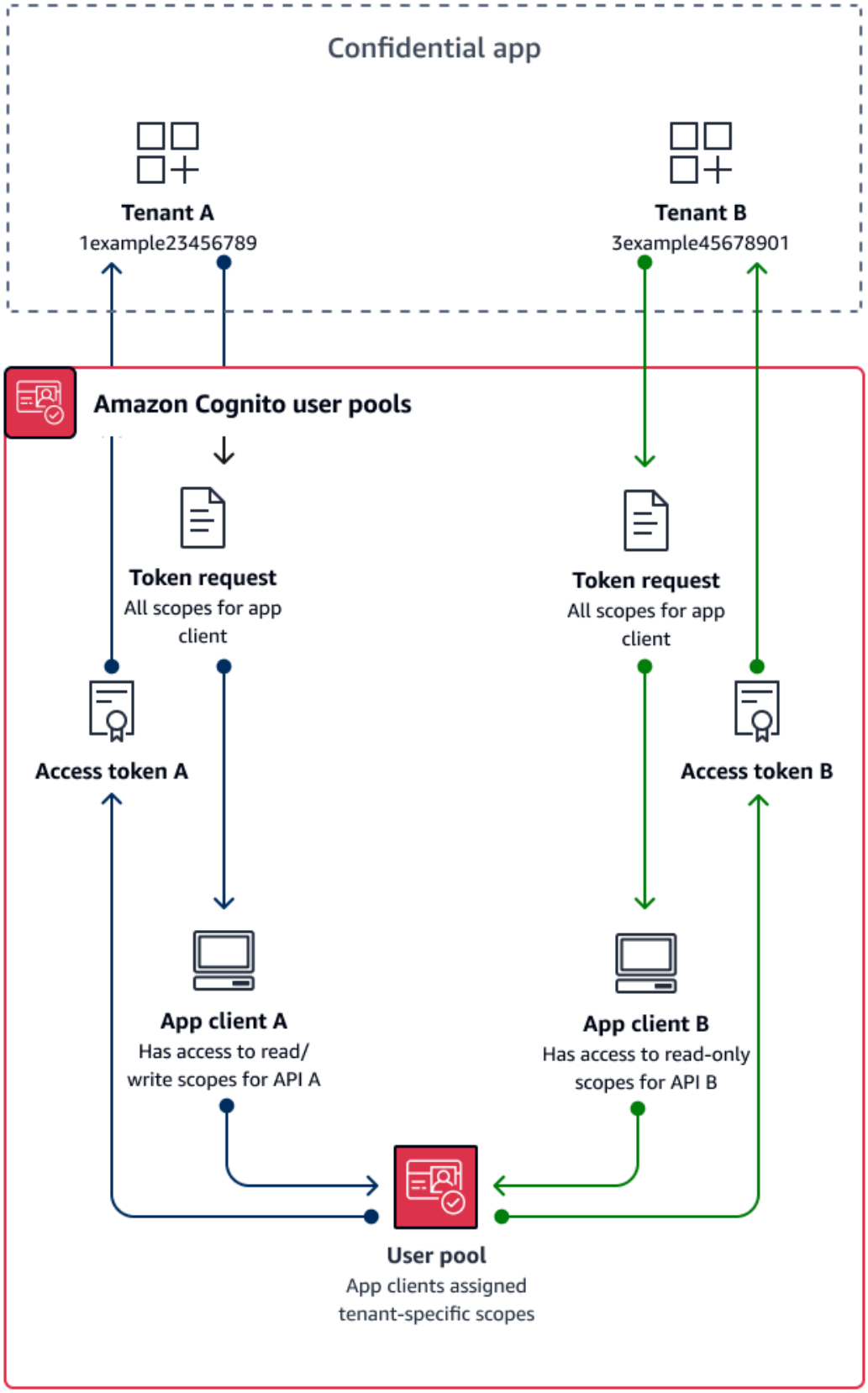
Level of effort

Because custom-attribute multi-tenancy transfers the duty of tenant-based authorization decisions on your app, the level of effort tends to be high. If you're already well-versed in a client configuration that parses OIDC claims, or in Amazon Verified Permissions, this approach might require the lowest level of effort.

Custom scope multi-tenancy best practices

Amazon Cognito supports custom OAuth 2.0 scopes for resource servers. You can implement app client multi-tenancy in users pools for machine-to-machine (M2M) authorization models with custom scopes. Scope-based multi-tenancy reduces the effort required to implement M2M multi-tenancy by defining access in your app client or application configuration.

The following diagram illustrates one option for custom scope multi-tenancy. It shows each tenant with a dedicated app client that has access to relevant scopes in a user pool.


Confidential app
Tenant A
1example23456789
Tenant B
3example45678901
Amazon Cognito user pools
Token request
All scopes for app client
Token request
All scopes for app client
Access token A
Access token B
App client A
Has access to read/ write scopes for API A
App client B
Has access to read-only scopes for API B
User pool
App clients assigned tenant-specific scopes

When to implement custom-scope multi-tenancy

When your usage is M2M authorization with client credentials in a confidential client. As a best practice, create resource servers that are exclusive to an app client. Custom scope multi-tenancy can be *request-dependent* or *client-dependent*.

Request-dependent

Implement application logic to request only the scopes that match the requirements of your tenant. For example, an app client might be able to issue read and write access to API A and API B, but tenant application A requests only the read scope for API A and the scope that indicates tenancy. This model allows for more complex combinations of shared scopes between tenants.

Client-dependent

Request all scopes assigned to an app client in your authorization requests. To do this, omit the `scope` request parameter in your request to the Token endpoint. If you obtain M2M access tokens with the GetClientToken API operation, omit the `Scopes` parameter for the same result. This model allows for app clients to store the access indicators that you want to add to your custom scopes.

In either case, your applications receive access tokens with scopes that indicate their privileges for data sources that they depend on. Scopes can also present other information to your application:

- Designate tenancy
- Contribute to request logging
- Indicate the APIs that the application is authorized to query
- Inform initial checks for active customers.

Level of effort

Custom-scope multi-tenancy requires a varying level of effort relative to the scale of your application. You must devise application logic that allows your applications to parse access tokens and make the appropriate API requests.

For example, a resource server scope comes in the format `[resource server identifier]/[name]`. The resource server identifier is unlikely to be relevant to the authorization decision from the tenant scope, requiring the scope name to be consistently parsed.

Example resource

The following AWS CloudFormation template creates a user pool for custom-scope multi-tenancy with one resource server and app client.

```
AWSTemplateFormatVersion: "2010-09-09"
Description: A sample template illustrating scope-based multi-tenancy
Resources:
  MyUserPool:
    Type: "AWS::Cognito::UserPool"
  MyUserPoolDomain:
    Type: AWS::Cognito::UserPoolDomain
    Properties:
      UserPoolId: !Ref MyUserPool
      # Note that the value for "Domain" must be unique across all of AWS.
      # In production, you may want to consider using a custom domain.
      # See: https://docs.aws.amazon.com/cognito/latest/developerguide/cognito-user-
pools-add-custom-domain.html#cognito-user-pools-add-custom-domain-adding
      Domain: !Sub "example-userpool-domain-${AWS::AccountId}"
  MyUserPoolResourceServer:
    Type: "AWS::Cognito::UserPoolResourceServer"
    Properties:
      Identifier: resource1
      Name: resource1
      Scopes:
        - ScopeDescription: Read-only access
          ScopeName: readScope
      UserPoolId: !Ref MyUserPool
  MyUserPoolTenantBatch1ResourceServer:
    Type: "AWS::Cognito::UserPoolResourceServer"
    Properties:
      Identifier: TenantBatch1
      Name: TenantBatch1
      Scopes:
        - ScopeDescription: tenant1 identifier
          ScopeName: tenant1
        - ScopeDescription: tenant2 identifier
          ScopeName: tenant2
      UserPoolId: !Ref MyUserPool
  MyUserPoolClientTenant1:
    Type: "AWS::Cognito::UserPoolClient"
    Properties:
      AllowedOAuthFlows:
```

```
        - client_credentials
      AllowedOAuthFlowsUserPoolClient: true
      AllowedOAuthScopes:
        - !Sub "${MyUserPoolTenantBatch1ResourceServer}/tenant1"
        - !Sub "${MyUserPoolResourceServer}/readScope"
      GenerateSecret: true
      UserPoolId: !Ref MyUserPool
Outputs:
  UserPoolClientId:
    Description: User pool client ID
    Value: !Ref MyUserPoolClientTenant1
  UserPoolDomain:
    Description: User pool domain
    Value: !Sub "https://${MyUserPoolDomain}.auth.${AWS::Region}.amazoncognito.com"
```

Multi-tenancy security recommendations

To help make your application more secure, we recommend the following:

- Validate tenancy in your app with Amazon Verified Permissions. Build policies that examine user pool, app client, group, or custom-attribute entitlement before you permit a user's request in your application. AWS created Verified Permissions identity sources with Amazon Cognito user pools in mind. Verified Permissions has additional guidance for multi-tenancy management.
- Use only a verified email address to authorize user access to a tenant based on domain match. Do not trust email addresses and phone numbers unless your app verifies them, or the external IdP gives a proof of verification. For more details on setting these permissions, see Attribute Permissions and Scopes.
- Use *immutable*, or read-only, custom attributes for the user profile attributes that identify tenants. You can only set the value of immutable attributes when you create a user or a user signs up in your user pool. Also, give app clients read-only access to the attributes.
- Use 1:1 mapping between a tenant's external IdP and application client to prevent unauthorized cross-tenant access. A user who has been authenticated by an external IdP, and who has a valid Amazon Cognito session cookie, can access other tenant apps that trust the same IdP.
- When you implement tenant-matching and authorization logic in your application, restrict users so that they can't modify the criteria that authorize user access to the tenants. Also, if an external IdP is being used for federation, restrict tenant identity provider administrators so that they can't modify user access.

Integrating Amazon Cognito authentication and authorization with web and mobile apps

Implementation of Amazon Cognito is a mix of AWS Management Console or AWS SDK administrative tools, and SDK libraries in applications. The Amazon Cognito console is the visual interface for setup and management of your Amazon Cognito user pools and identity pools.

The lowest-effort integration you can create with Amazon Cognito user pools is with managed login. Managed login is a ready-to-use web-based sign-in application for quick testing and deployment of Amazon Cognito user pools. User pool authentication with managed login requires OpenID Connect (OIDC) libraries that direct users to hosted sign-in pages. In this series of user-interactive and redirect web endpoints, Amazon Cognito handles the flow of authentication, including third-party sign-in, multi-factor authentication (MFA), and choosing an authentication flow. Your application only has to process the authentication outcome that Amazon Cognito returns in the response.

You can also add an AWS SDK to your application, custom-build authentication interfaces, and invoke API operations for authentication and authorization of your users. AWS Amplify is an AWS service for building full-stack applications, with Amazon Cognito authentication in the back end.

For example, your app might invoke managed login for user sign-in, then call the token endpoint from your app code to exchange your user's authorization code for tokens. Then your app must interpret and store your user's tokens, and present them in the appropriate context for authentication and authorization. Amplify adds guided integration tools with built-in functions for these processes.

You can also build your Amazon Cognito resources entirely in code. Identity pools don't have the same managed authentication options as user pools—for access to AWS credentials in your applications, implement identity pools operations in imported SDK modules. To get started with your own custom-built application code, visit the Amazon Cognito code examples for AWS SDKs. For integration with the Amazon Cognito as an OpenID Connect identity provider, use OpenID Connect developer tools.

Before you use Amazon Cognito authentication and authorization, choose an app platform and prepare your code to integrate with the service. For available platforms for AWS SDKs, see Authentication with AWS SDKs. The AWS CLI is a command-line SDK for Amazon Cognito and other

AWS services, and is a valuable place to begin to familiarize yourself with Amazon Cognito API operations and their syntax.

> **Note**
>
> Some components of Amazon Cognito can be configured only with the API. For example, you can only set a user pool custom SMS or email sender Lambda trigger with a request that updates the `LambdaConfig` property of the UserPool class in a CreateUserPool or UpdateUserPool API request.

The Amazon Cognito user pools API shares its namespace with several classes of API operations. One class configures user pools and their processes, identity providers and users. Another includes unauthenticated operations for your users in a public client to sign in, sign out, and manage their profiles. The final class of API operations performs user operations that you authorize with your own AWS credentials in a confidential server-side client. You must know your intended app architecture before you begin to implement app code. For more information, see Understanding API, OIDC, and managed login pages authentication.

Topics

- Authentication with AWS Amplify
- Authentication with AWS SDKs
- How authentication works with Amazon Cognito
- Using this service with an AWS SDK
- Authorization with Amazon Verified Permissions

Authentication with AWS Amplify

AWS Amplify is a complete solution for building web and mobile applications. With Amplify, you can connect to existing resources with the Amplify libraries, or you can create and configure new resources with the Amplify command line interface (CLI). Amplify also has connected UI components like Authenticator for setup and customization of the sign-in and sign-up experience in your app.

To use Amplify authentication features in your front-end app, see the following documentation by platform.

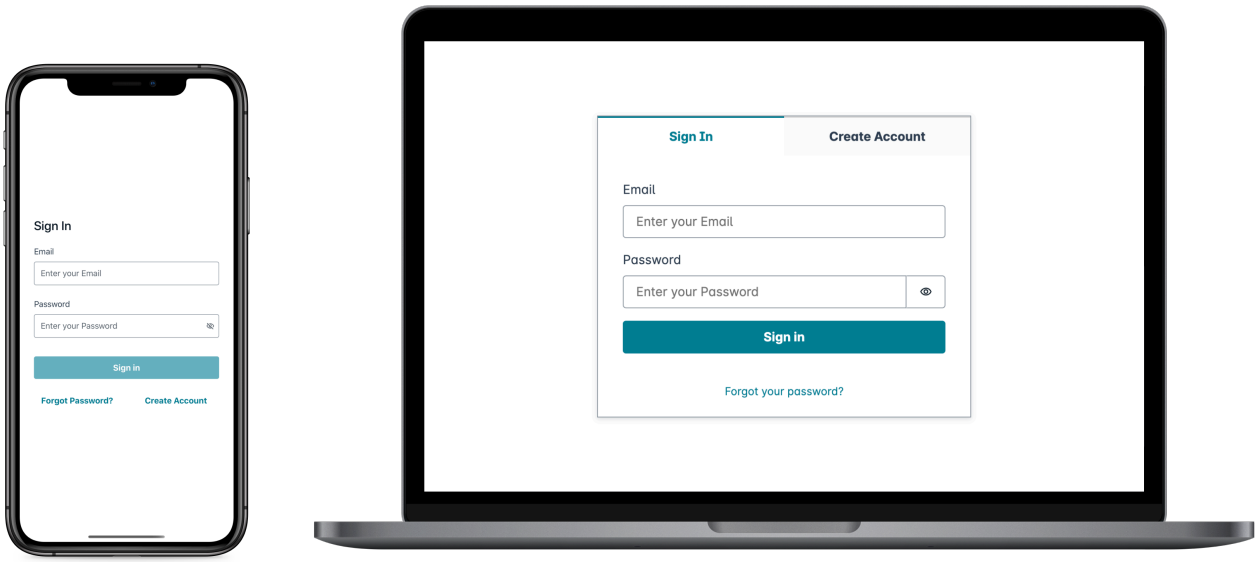
- Amplify authentication for React
- Amplify authentication for React Native
- Amplify authentication for Swift (iOS)
- Amplify authentication for Android
- Amplify authentication for Flutter

The Amplify libraries are open source and are available on GitHub. To learn more about how Amplify Auth implements Amazon Cognito authentication, visit the following libraries.

- amplify-js
- amplify-swift
- amplify-flutter
- amplify-android

Creating a user interface (UI) with Amplify

User pool managed login can fulfill the essential needs of an authentication front-end for a web or mobile app. To customize your user interface (UI) beyond the parameters that managed login accommodates, custom-build an application. Amplify UI is a customizable collection of front-end components in a variety of languages.



To get started with your custom authentication component, visit the following documentation for the Authenticator component.

- Authenticator for Android
- Authenticator for Angular
- Authenticator for Flutter
- Authenticator for React
- Authenticator for React Native
- Authenticator for Swift
- Authenticator for Vue

Authentication with AWS SDKs

To use a secure backend to build your own identity microservice that interacts with Amazon Cognito, connect to the Amazon Cognito user pools and Amazon Cognito identity pools API with an AWS SDK in the language of your choice.

For details on each API operation, see the Amazon Cognito user pools API Reference and the Amazon Cognito API Reference. These documents contain **See also** sections with resources for using a variety of SDKs in supported platforms.

- AWS Command Line Interface
- AWS SDK for .NET
- AWS SDK for C++
- AWS SDK for Go
- AWS SDK for Java V2
- AWS SDK for JavaScript
- AWS SDK for PHP V3
- AWS SDK for Python
- AWS SDK for Ruby V3

How authentication works with Amazon Cognito

When your customer signs in to an Amazon Cognito user pool, your application receives JSON web tokens (JWTs).

When your customer signs in to an identity pool, either with a user pool token or another provider, your application receives temporary AWS credentials.

With user pool sign-in, you can implement authentication and authorization entirely with an AWS SDK. If you don't want to build your own user interface (UI) components, you can invoke a prebuilt web UI (managed login) or the sign-in page for your third-party identity provider (IdP).

This topic is an overview of some of the ways that your application can interact with Amazon Cognito to authenticate with ID tokens, authorize with access tokens, and access AWS services with identity pool credentials.

Topics

- User pool authentication with managed login
- User pool API authentication and authorization with an AWS SDK
- User pool authentication with a third-party identity provider
- Identity pool authentication

User pool authentication with managed login

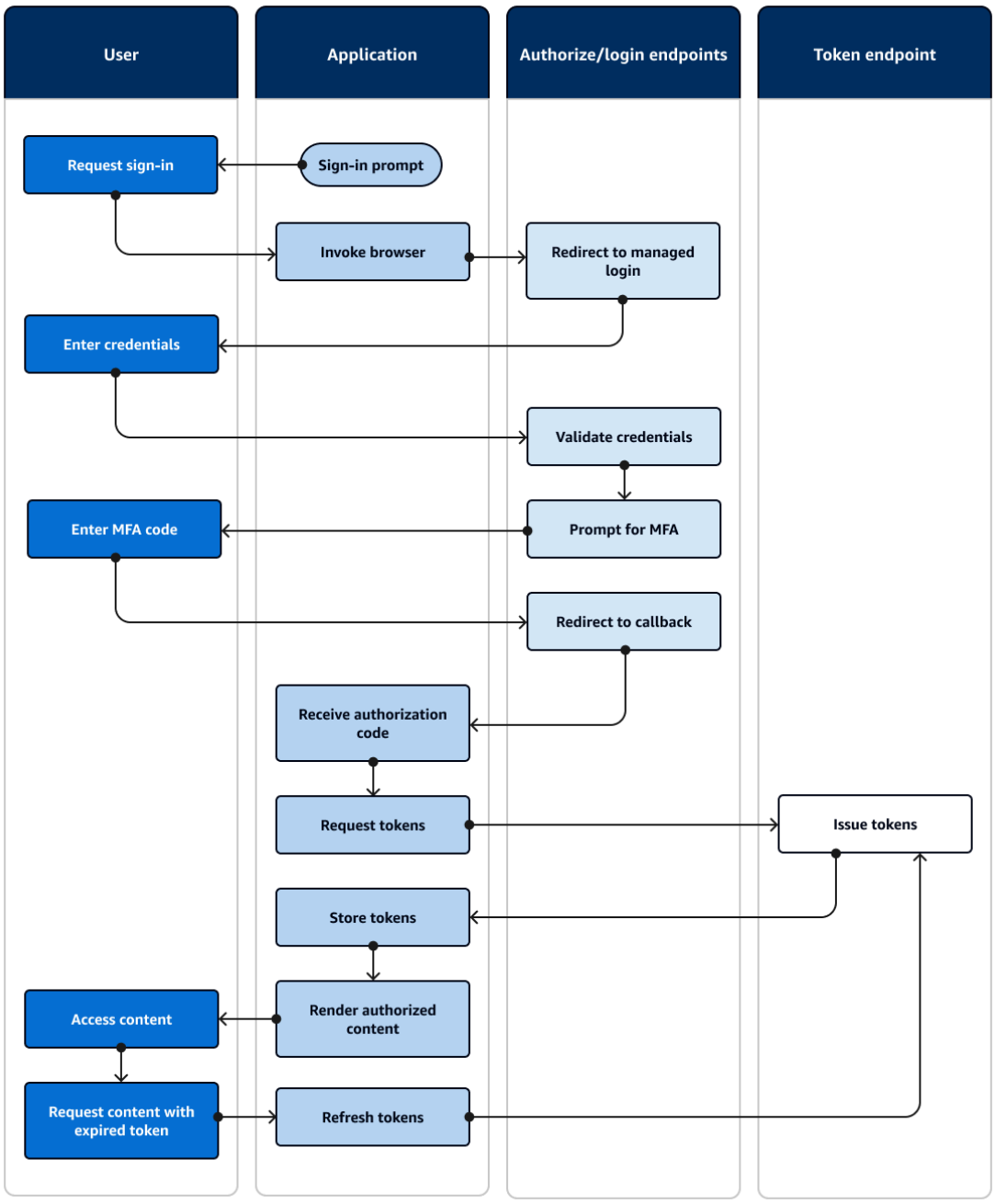
Managed login is a website that is linked to your user pool and app client. It can perform sign-in, sign-up, and password-reset operations for your users. An application with a managed login component for authentication can require less developer effort to implement. An application can skip UI components for authentication and invoke managed login webpages in the user's browser.

Applications collect users' JWTs with a web or app redirect location. Applications that implement managed login can connect to user pools for authentication as if they were an OpenID Connect (OIDC) IdP.

Managed login fits the model where applications require the authentication services of an OIDC authorization server, but don't immediately require features like custom authentication, identity pools integration, or user attribute self-service. When you want to use some of these advanced options, you can implement them with a user pools component for an SDK.

Managed login and third-party IdP authentication models, with a primary reliance on OIDC implementation, are best for advanced authorization models with OAuth 2.0 scopes.

The following diagram illustrates a typical sign-in session for managed login authentication.



Managed login authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. The application directs the user to a sign-in prompt in the managed login pages of your user pool domain.
4. They enter their username and password.

5. The user pool validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The managed login page prompts the user to enter an MFA code.
7. The user enters their MFA code.
8. Your user pool redirects the user to the application URL.
9. The application collects the authorization code from the URL request parameter that managed login appended to the callback URL.
10. The application requests tokens with the authorization code.
11. The token endpoint returns JWTs to the application.
12. The application decodes, validates, and stores or caches the user's JWTs.
13. The application displays the requested access-controlled component.
14. The user views their content.
15. Later, the user's access token has expired, and they request to view an access-controlled component.
16. The application determines that the user's session should persist. It requests new tokens from the token endpoint with the refresh token.

Variants and customization

You can customize the look and feel of your managed login pages with the branding editor for your entire user pool, or at the level of any app client. You can also configure app clients with their own identity providers, scopes, access to user attributes, and advanced security configuration.

Related resources

- User pool managed login
- Scopes, M2M, and resource servers
- User pool endpoints and managed login reference

User pool API authentication and authorization with an AWS SDK

AWS has developed components for Amazon Cognito user pools, or *Amazon Cognito identity provider*, in a variety of developer frameworks. The methods built into these SDKs call the Amazon Cognito user pools API. The same user pools API namespace has operations for configuration of

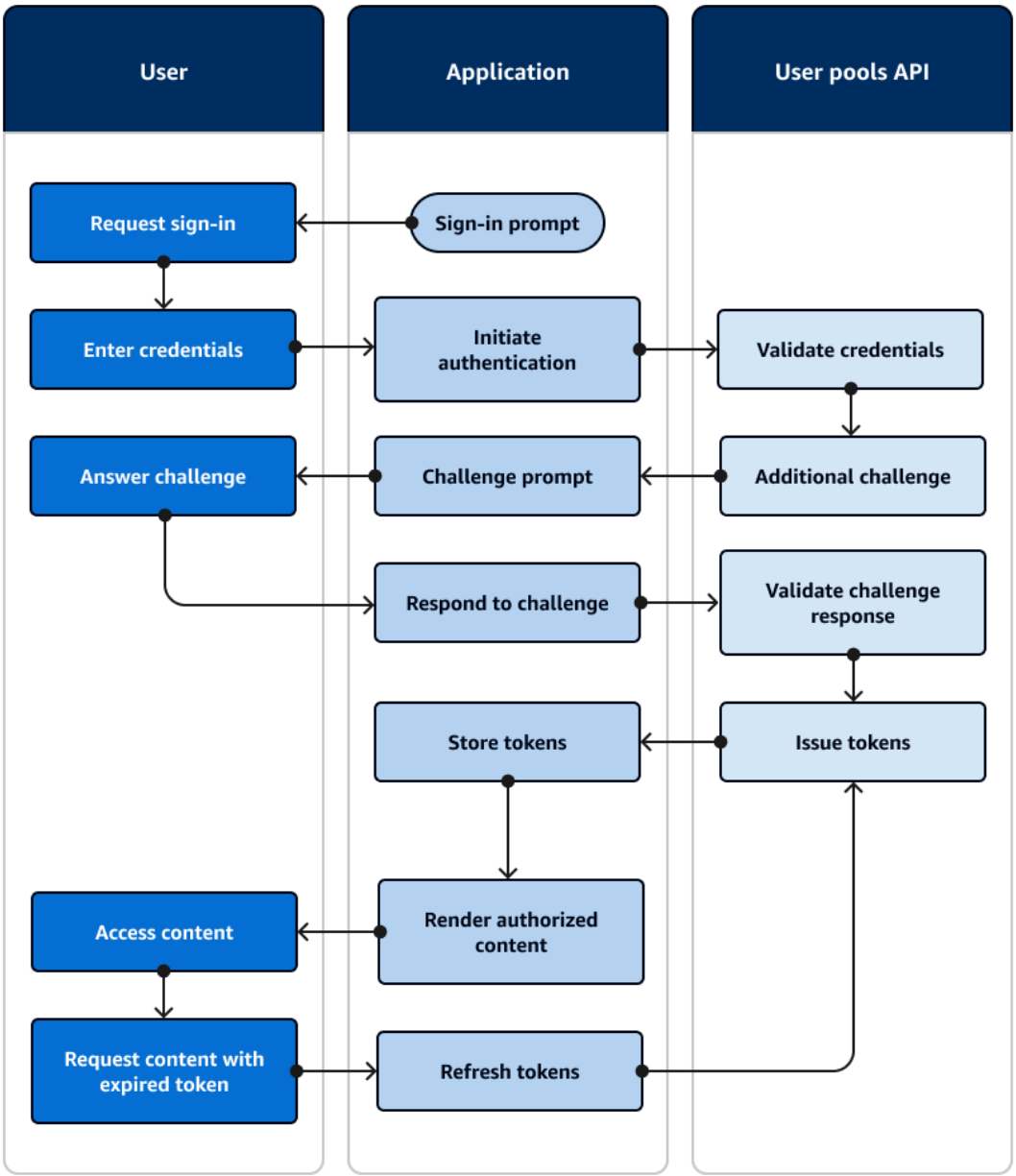
user pools and for user authentication. For a more thorough overview, see Understanding API, OIDC, and managed login pages authentication.

API authentication fits the model where your applications have existing UI components and primarily rely on the user pool as a user directory. This design adds Amazon Cognito as a component within a larger application. It requires programmatic logic to handle complex chains of challenge and response.

This application doesn't need to implement a full OpenID Connect (OIDC) relying party implementation. Instead, it has the ability to decode and use JWTs. When you want access to the full set of user pool features for local users, build your authentication with the Amazon Cognito SDK in your development environment.

API authentication with custom OAuth scopes is less oriented toward external API authorization. To add custom scopes to an access token from API authentication, modify the token at runtime with a Pre token generation Lambda trigger.

The following diagram illustrates a typical sign-in session for API authentication.



API authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. They enter their username and password.
4. The application invokes the method that makes an InitiateAuth API request. The request passes the user's credentials to a user pool.

5. The user pool validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The user pool responds with a challenge that requests an MFA code.
7. The application generates a prompt that collects the MFA code from the user.
8. The application invokes the method that makes a RespondToAuthChallenge API request. The request passes the user's MFA code.
9. The user pool validates the user's MFA code.
10. The user pool responds with the user's JWTs.
11. The application decodes, validates, and stores or caches the user's JWTs.
12. The application displays the requested access-controlled component.
13. The user views their content.
14. Later, the user's access token has expired, and they request to view an access-controlled component.
15. The application determines that the user's session should persist. It invokes the InitiateAuth method again with the refresh token and retrieves new tokens.

Variants and customization

You can augment this flow with additional challenges—for example, your own custom authentication challenges. You can automatically restrict access for users whose passwords have been compromised, or whose unexpected sign-in characteristics might indicate a malicious sign-in attempt. This flow looks much the same for operations to sign up, update user attributes, and reset passwords. Most of these flows have duplicate public (client-side) and confidential (server-side) API operations.

Related resources

- Amazon Cognito user pools API
- Create a new application in the Amazon Cognito console
- Integrating Amazon Cognito authentication and authorization with web and mobile apps
- Understanding API, OIDC, and managed login pages authentication

User pool authentication with a third-party identity provider

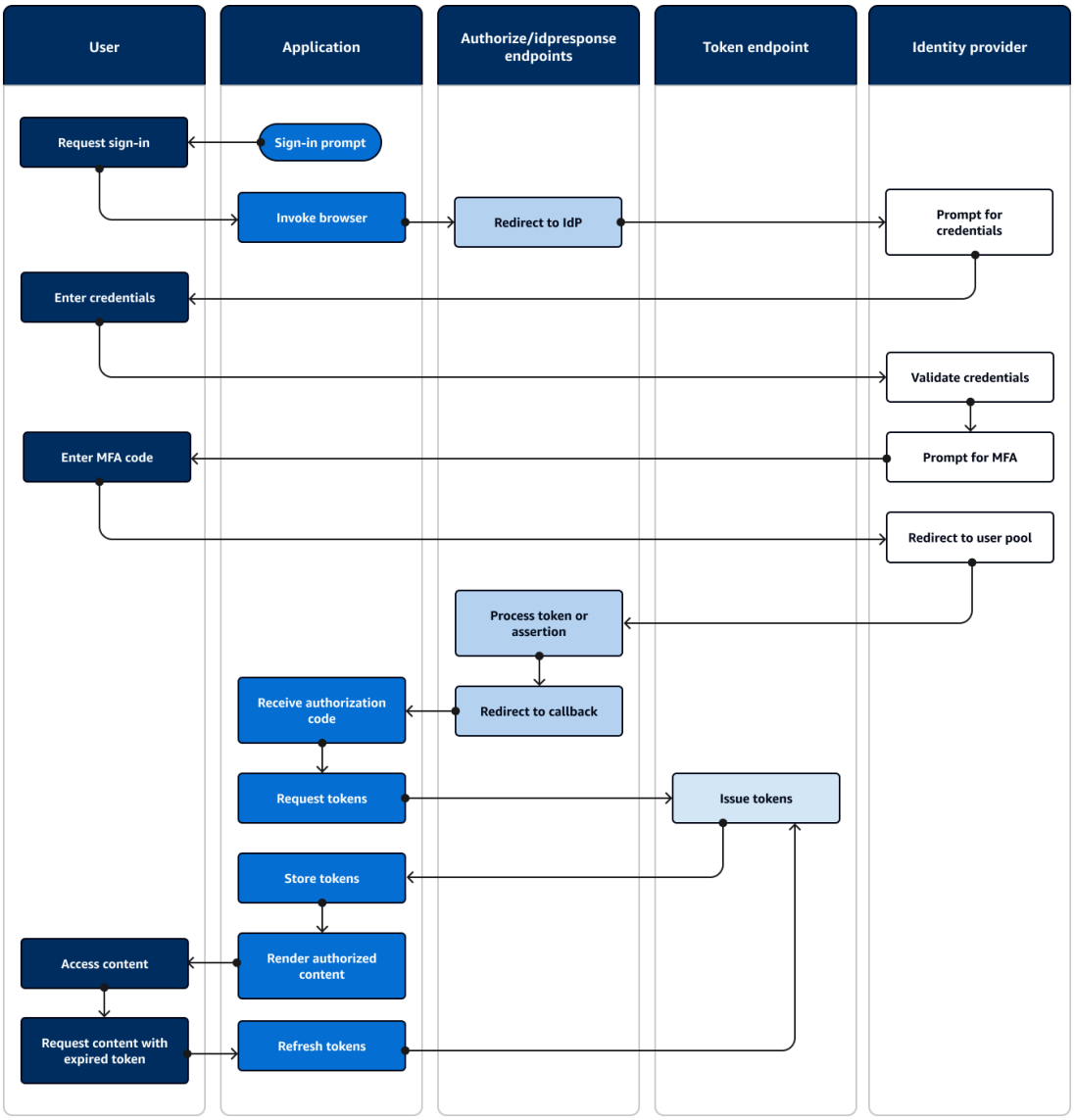
Sign-in with an external identity provider (IdP), or *federated authentication*, is a similar model to managed login. Your application is an OIDC relying party to your user pool, while your user pool serves as a passthrough to an IdP. The IdP can be a consumer user directory like Facebook or Google, or it can be a SAML 2.0 or OIDC enterprise directory like Azure.

Instead of managed login in the user's browser, your application invokes a redirect endpoint on the user pool authorization server. From the user's view, they choose the sign-in button in your application. Then their IdP prompts them to sign in. Like with managed login authentication, an application collects JWTs at a redirect location in the app.

Authentication with a third-party IdP fits a model where users might not want to come up with a new password when they sign up for your application. Third-party authentication can be added with low effort to an application that's implemented managed login authentication. In effect, managed login and third-party IdPs produce a consistent authentication outcome from minor variations in what you invoke in users' browsers.

Like managed login authentication, federated authentication is best for advanced authorization models with OAuth 2.0 scopes.

The following diagram illustrates a typical sign-in session for federated authentication.



Federated authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. The application directs the user to a sign-in prompt with their IdP.
4. They enter their username and password.
5. The IdP validates the user's credentials and determines that the user has activated multi-factor authentication (MFA).
6. The IdP prompts the user to enter an MFA code.
7. The user enters their MFA code.

8. The IdP redirects the user to the user pool with a SAML response or an authorization code.
9. If the user passed an authorization code, the user pool silently exchanges the code for IdP tokens. The user pool validates the IdP tokens and redirects the user to the application with a new authorization code.
10. The application collects the authorization code from the URL request parameter that the user pool appended to the callback URL.
11. The application requests tokens with the authorization code.
12. The token endpoint returns JWTs to the application.
13. The application decodes, validates, and stores or caches the user's JWTs.
14. The application displays the requested access-controlled component.
15. The user views their content.
16. Later, the user's access token has expired, and they request to view an access-controlled component.
17. The application determines that the user's session should persist. It requests new tokens from the token endpoint with the refresh token.

Variants and customization

You can initiate federated authentication in managed login, where users can choose from a list of IdPs that you assigned to your app client. Managed login can also prompt for an email address and automatically route a user's request to the corresponding SAML IdP. Authentication with a third-party identity provider doesn't *require* user interaction with managed login. Your application can add a request parameter to a user's authorization server request and cause the user to silently redirect to their IdP sign-in page.

Related resources

- User pool sign-in with third party identity providers
- Scopes, M2M, and resource servers
- User pool endpoints and managed login reference

Identity pool authentication

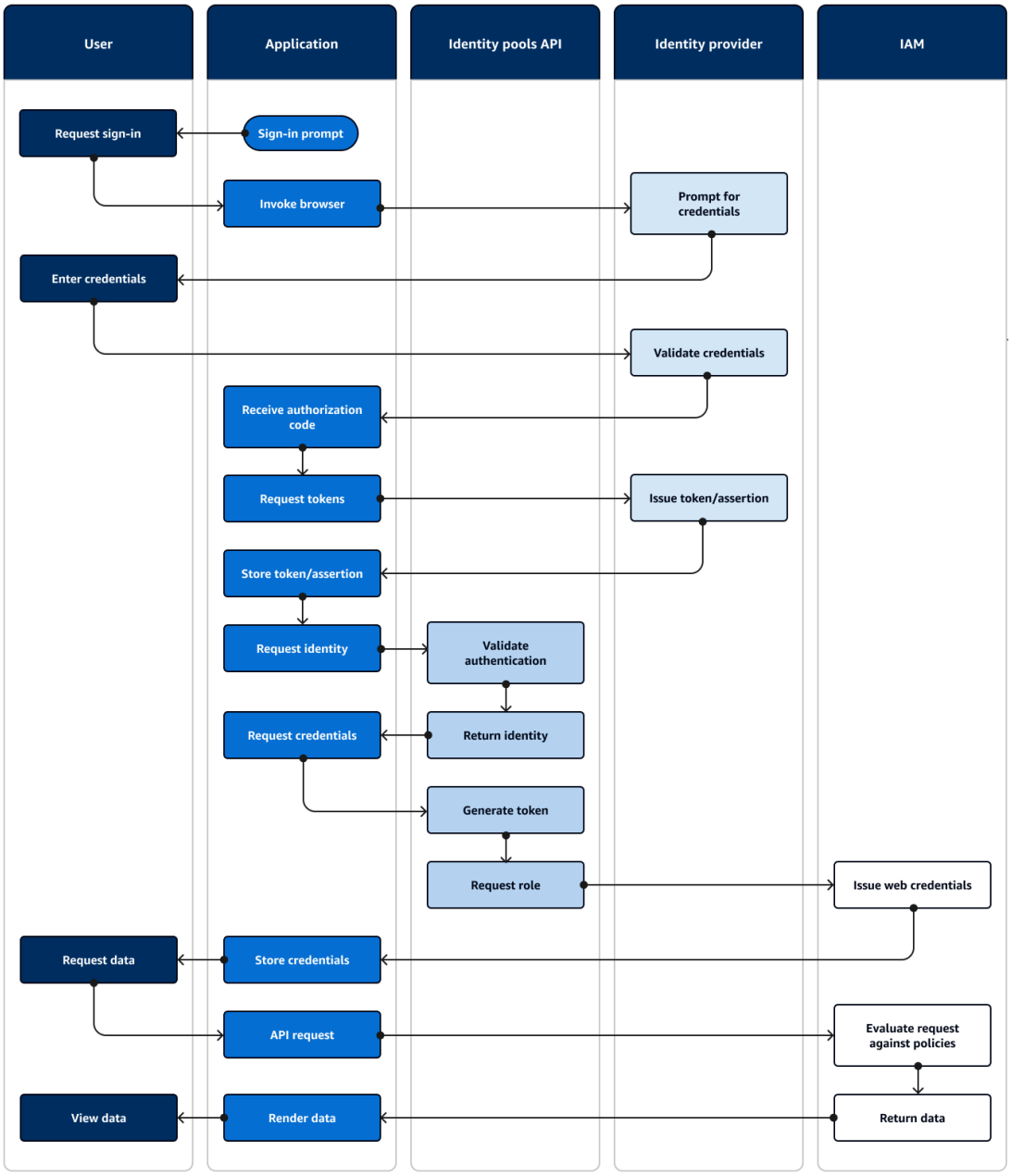
An identity pool is a component for your application that is distinct from a user pool in function, API namespace, and SDK model. Where user pools offer token-based authentication and authorization, identity pools offer authorization for AWS Identity and Access Management (IAM).

You can assign a set of IdPs to identity pools and sign in users with them. User pools are closely integrated as identity pool IdPs and give identity pools the most options for access control. At the same time, there is a wide selection of authentication options for identity pools. User pools join SAML, OIDC, social, developer, and guest identity sources as routes to temporary AWS credentials from identity pools.

Authentication with an identity pool is external—it follows one of the previously illustrated user pool flows, or a flow that you develop independently with another IdP. After your application performs initial authentication, it passes the proof to an identity pool and receives a temporary session in return.

Authentication with an identity pool fits a model where you enforce the access control for application assets and data in AWS services with IAM authorization. Like with API authentication in user pools, a successful application includes AWS SDKs for each of the services that you want to access for your users' benefit. AWS SDKs apply the credentials from identity pool authentication as signatures to API requests.

The following diagram illustrates a typical sign-in session for identity pool authentication with an IdP.



Identity pool authentication flow

1. A user accesses your application.
2. They select a "Sign in" link.
3. The application directs the user to a sign-in prompt with their IdP.
4. They enter their username and password.
5. The IdP validates the user's credentials.

6. The IdP redirects the user to the application with a SAML response or an authorization code.
7. If the user passed an authorization code, the application exchanges the code for IdP tokens.
8. The application decodes, validates, and stores or caches the user's JWTs or assertion.
9. The application invokes the method that makes a GetId API request. It passes the user's token or assertion and requests an identity ID.
10. The identity pool validates the token or assertion against configured identity providers.
11. The identity pool returns an identity ID.
12. The application invokes the method that makes a GetCredentialsForIdentity API request. It passes the user's token or assertions and requests an IAM role.
13. The identity pool generates a new JWT. The new JWT contains claims that request an IAM role. The identity pool determines the role based on the user's request and the role-selection criteria in the identity pool configuration for the IdP.
14. AWS Security Token Service (AWS STS) responds to the AssumeRoleWithWebIdentity request from the identity pool. The response contains API credentials for a temporary session with an IAM role.
15. The application stores the session credentials.
16. The user takes an action in the app that requires access-protected resources in AWS.
17. The application applies the temporary credentials as signatures to API requests for the required AWS services.
18. IAM evaluates the policies attached to the role in the credentials. It compares them to the request.
19. The AWS service returns the requested data.
20. The application renders the data in the user's interface.
21. The user views the data.

Variants and customization

To visualize authentication with a user pool, insert one of the previous user-pool overviews after the **Issue token/assertion** step. Developer authentication replaces all steps before **Request identity** with a request signed by developer credentials. Guest authentication also skips straight to **Request identity**, doesn't validate authentication, and returns credentials for a limited-access IAM role.

Related resources

- Amazon Cognito identity pools
- User IAM roles
- Identity pools authentication flow

Using this service with an AWS SDK

AWS software development kits (SDKs) are available for many popular programming languages. Each SDK provides an API, code examples, and documentation that make it easier for developers to build applications in their preferred language.

Authorization with Amazon Verified Permissions

Amazon Verified Permissions is an authorization service for the applications that you build. When you add an Amazon Cognito user pool as an identity source, your app can pass user pool access or identity (ID) tokens to Verified Permissions for an allow or deny decision. Verified Permissions considers your user's properties and request context based on policies that you write in Cedar Policy Language. The request context can include an identifier for the document, image, or other resource they requested, and the action that your user wants to take on the resource.

Your app can provide your user's identity or access tokens to Verified Permissions in IsAuthorizedWithToken or BatchIsAuthorizedWithToken API requests. These API operations accept your users as a `Principal` and make authorization decisions for the `Action` on the `Resource` that they want to access. Additional custom `Context` can contribute to a detailed access decision.

When your app presents a token in an `IsAuthorizedWithToken` API request, Verified Permissions performs the following validations.

1. Your user pool is a configured Verified Permissions identity source for the requested policy store.
2. The `client_id` or aud claim, in your access or identity token respectively, matches a user pool app client ID that you provided to Verified Permissions. To verify this claim, you must configure client ID validation in your Verified Permissions identity source.
3. Your token isn't expired.
4. The value of the `token_use` claim in your token matches the parameters that you passed to `IsAuthorizedWithToken`. The `token_use` claim must be `access` if you passed it to the `accessToken` parameter, and `id` if you passed it to the `identityToken` parameter.

5. The signature in your token comes from the published JSON web keys (JWKs) of your user pool. You can view your JWKs at `https://cognito-idp.Region.amazonaws.com/your user pool ID/.well-known/jwks.json`.

Revoked tokens and deleted users

Verified Permissions only validates the information it knows from your identity source and from the expiration time of your user's token. Verified Permissions doesn't check for token revocation or user existence. If you revoked your user's token or deleted your user's profile from your user pool, Verified Permissions still considers the token valid until it expires.

Policy evaluation

Configure your user pool as an identity source for your policy store. Configure your app to submit your users' tokens in requests to Verified Permissions. For each request, Verified Permissions compares the claims in the token to a policy. A Verified Permissions policy is like an IAM policy in AWS. It declares a *principal*, *resource*, and *action*. Verified Permissions responds to your request with `Allow` if it matches an allowed action and doesn't match an explicit `Deny` action; otherwise, it responds with `Deny`. For more information, see Amazon Verified Permissions policies in the *Amazon Verified Permissions User Guide*.

Customizing tokens

To change, add, and remove the user claims that you want to present to Verified Permissions, customize the content in your access and identity tokens with a Pre token generation Lambda trigger. With a pre token generation trigger, you can add and modify claims in your tokens. For example, you can query a database for additional user attributes and encode them into your ID token.

> **Note**
>
> Because of the way that Verified Permissions processes claims, don't add claims named `cognito`, `dev`, or `custom` in your pre token generation function. When you present these reserved claim prefixes not in colon-delimited format like `cognito:username` but as full claim names, your authorization requests fail.

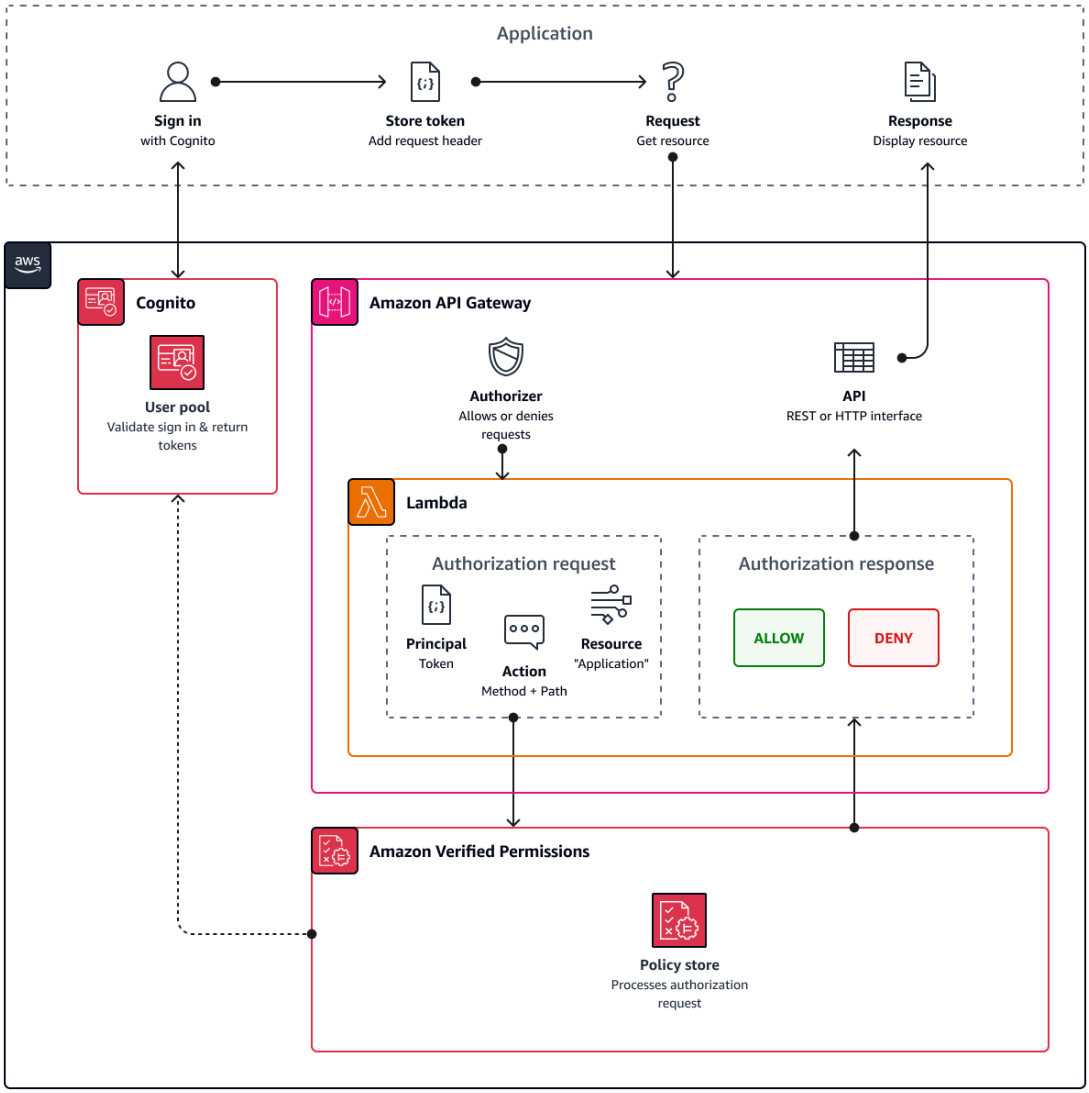
Additional resources

- Mapping Amazon Cognito tokens to Verified Permissions schema
- Authorize API Gateway APIs using Amazon Verified Permissions and Amazon Cognito
- Workshop: Authentication and authorization with Amazon Cognito and Verified Permissions

API authorization with Verified Permissions

Your ID or access tokens can authorize requests to back-end Amazon API Gateway REST APIs with Verified Permissions. You can create a policy store with immediate links to your user pool and API. With the Set up with API Gateway and an identity source starting option, Verified Permissions adds a user pool identity source to the policy store, and a Lambda authorizer to the API. When your application passes a user pool bearer token to the API, the Lambda authorizer invokes Verified Permissions. The authorizer passes the token as a principal and the request path and method as an action.

The following diagram illustrates the authorization flow for an API Gateway API with Verified Permissions. For a detailed breakdown, see API-linked policy stores in the Amazon Verified Permissions User Guide.



Verified Permissions structures API authorization around user pool groups. Because both ID and access tokens include a `cognito:groups` claim, your policy store can manage role-based access control (RBAC) for your APIs in a variety of application contexts.

Choosing policy store settings

When you configure an identity source on a policy store, you must choose whether you want to process access or ID tokens. This decision is significant to the way that your policy engine operates. ID tokens contain user attributes. Access tokens contain user access-control information: OAuth scopes. Although both token types have group-membership information, we generally recommend the access token for RBAC with a Verified Permissions policy store. The access token adds to group

membership with scopes that can contribute to the authorization decision. The claims in an access token become context in the authorization request.

You must also configure the user and group entity types when you configure a user pool as an identity source. Entity types are principal, action, and resource identifiers that you can reference in Verified Permissions policies. Entities in policy stores can have a *membership* relationship, where one entity can be a member of a *parent* entity. With membership, you can reference principal groups, action groups, and resource groups. In the case of user pool groups, the user entity type that you specify must be a member of the group entity type. When you set up an API-linked policy store or follow **Guided setup** in the Verified Permissions console, your policy store automatically has this parent-member relationship.

The ID token can combine RBAC with attribute-based access control (ABAC). After you create an API-linked policy store, you can enhance your policies with user attributes *and* group membership. The attribute claims in an ID token become principal attributes in the authorization request. Your policies can make authorization decisions based on principal attributes.

You can also configure a policy store to accept tokens with an `aud` or `client_id` claim that matches a list of acceptable app clients that you provide.

Example policy for role-based API authorization

The following example policy was created by the setup of a Verified Permissions policy store for a PetStore example REST API.

```
permit(
  principal in PetStore::UserGroup::"us-east-1_EXAMPLE|MyGroup",
  action in [ PetStore::Action::"get /pets", PetStore::Action::"get /pets/{petId}" ],
  resource
  );
```

Verified Permissions returns an `Allow` decision to the authorization request from your application when:

1. Your application passed an ID or access token in an `Authorization` header as a bearer token.
2. Your application passed a token with a `cognito:groups` claim that contains the string `MyGroup`.
3. Your application made an `HTTP GET` request to, for example, `https://myapi.example.com/pets` or `https://myapi.example.com/pets/scrappy`.

Example policy for an Amazon Cognito user

Your user pool can also generate authorization requests to Verified Permissions in conditions other than API requests. You can submit any access control decisions in your application to your policy store. For example, you can supplement Amazon DynamoDB or Amazon S3 security with attribute-based access control before any requests transit the network, reducing quota use.

The following example uses the Cedar Policy Language to permit Finance users who authenticate with one user pool app client to read and write `example_image.png`. John, a user in your app, receives an ID token from your app client and passes it in a GET request to a URL that requires authorization, `https://example.com/images/example_image.png`. John's ID token has an `aud` claim of your user pool app client ID `1234567890example`. Your pre token generation Lambda function also inserted a new claim `costCenter` with a value, for John, of `Finance1234`.

```
permit (
   principal,
   actions in [ExampleCorp::Action::"readFile", "writeFile"],
   resource == ExampleCorp::Photo::"example_image.png"
)
when {
   principal.aud == "1234567890example" &&
   principal.custom.costCenter like "Finance*"
};
```

The following request body results in an `Allow` response.

```
{
   "accesstoken": "[John's ID token]",
   "action": {
      "actionId": "readFile",
      "actionType": "Action"
   },
   "resource": {
      "entityId": "example_image.png",
      "entityType": "Photo"
   }
}
```

When you want to specify a principal in a Verified Permissions policy, use the following format:

```
permit (
```

```
   principal == [Namespace]::[Entity]::"[user pool ID]|[user sub]",
   action,
   resource
);
```

The following is an example principal for a user in a user pool with ID `us-east-1_Example` with sub, or user ID, `973db890-092c-49e4-a9d0-912a4c0a20c7`.

```
principal == ExampleCorp::User::"us-east-1_Example|973db890-092c-49e4-
a9d0-912a4c0a20c7",
```

When you want to specify a user group in a Verified Permissions policy, use the following format:

```
permit (
   principal in [Namespace]::[Group Entity]::"[Group name]",
   action,
   resource
);
```

Attribute-based access control

Authorization with Verified Permissions for your apps, and the attributes for access control feature of Amazon Cognito identity pools for AWS credentials, are both forms of attribute-based access control (ABAC). The following is a comparison of the features of Verified Permissions and Amazon Cognito ABAC. In ABAC, a system examines the attributes of an entity and makes an authorization decision from conditions that you define.

Service	Process	Result
Amazon Verified Permissions	Returns an `Allow` or `Deny` decision from analysis of a user pool JWT.	Access to application resources succeeds or fails based on Cedar policy evaluation.
Amazon Cognito identity pools (attributes for access control)	Assigns session tags to your user based on their attributes. IAM policy conditions can check tags `Allow` or `Deny` user access to AWS services.	A tagged session with temporary AWS credentials for an IAM role.

User pool endpoints and managed login reference

Amazon Cognito has two models of user pool authentication: with the user pools API and with the OAuth 2.0 authorization server. Use the API when you want to retrieve OpenID Connect (OIDC) tokens with an AWS SDK in your application back end. Use the authorization server when you want to implement your user pool as an OIDC provider. The authorization server adds features like federated sign-in, API and M2M authorization with access token scopes, and managed login. You can use the API and OIDC models each on their own or together, configured at the user pool level or at the app client level. This section is a reference for the implementation of the OIDC model. For more information about the two authentication models, see Understanding API, OIDC, and managed login pages authentication.

Amazon Cognito activates the public webpages listed here when you assign a domain to your user pool. Your domain serves as a central access point for all of your app clients. They include managed login, where your users can sign up and sign in (Login endpoint), and sign out (Logout endpoint). For more information about these resources, see User pool managed login.

These pages also include the public web resources that allow your user pool to communicate with third-party SAML, OpenID Connect (OIDC) and OAuth 2.0 identity providers (IdPs). To sign in a user with a federated identity provider, your users must initiate a request to the interactive managed login Login endpoint or the OIDC Authorize endpoint. The Authorize endpoint redirects your users either to your managed login pages or your IdP sign-in page.

Your app can also sign in *local users* with the Amazon Cognito user pools API. A local user exists exclusively in your user pool directory without federation through an external IdP.

In addition to managed login, Amazon Cognito integrates with SDKs for Android, iOS, JavaScript, and more. The SDKs provide tools to perform user pool API operations with Amazon Cognito API service endpoints. For more information about service endpoints, see Amazon Cognito Identity endpoints and quotas.

Warning

Don't pin the end-entity or intermediate Transport Layer Security (TLS) certificates for Amazon Cognito domains. AWS manages all certificates for all of your user pool endpoints and prefix domains. The certificate authorities (CAs) in the chain of trust that supports Amazon Cognito certificates dynamically rotate and renew. When you pin your app to

an intermediate or leaf certificate, your app can fail without notice when AWS rotates certificates.
Instead, pin your application to all available Amazon root certificates. For more information, see best practices and recommendations at Certificate pinning in the *AWS Certificate Manager User Guide*.

For a list of service endpoints for the user pools API by AWS Region, see Service endpoints in the *AWS General Reference*.

Topics

- User-interactive managed login and classic hosted UI endpoints
- Identity provider and relying party endpoints
- OAuth 2.0 grants
- Using PKCE in authorization code grants
- Managed login and federation error responses

User-interactive managed login and classic hosted UI endpoints

Amazon Cognito activates the managed login endpoints in this section when you add a domain to your user pool. They are webpages where your users can complete the core authentication operations of a user pool. They include pages for password management, multi-factor authentication (MFA), and attribute verification.

The webpages that make up managed login are a front-end web application for interactive user sessions with your customers. Your app must invoke managed login in your users' browsers. Amazon Cognito doesn't support programmatic access to the webpages in this chapter. Those federation endpoints in the Identity provider and relying party endpoints that return a JSON response can be queried directly in your app code. The Authorize endpoint redirects either to managed login or to an IdP sign-in page and also must be opened in users' browsers.

All user pool endpoints accept traffic from IPv4 and IPv6 source IP addresses.

The topics in this guide describe frequently-used managed login and classic hosted UI endpoints in detail. The difference between managed login and the hosted UI is visible, not functional. Except for `/passkeys/add`, all paths are shared between the two versions of managed login branding.

Amazon Cognito makes the webpages that follow available when you assign a domain to your user pool.

Managed login endpoints

Endpoint URL	Description	How it's accessed
https://*Your user pool domain*/login	Signs in user pool local and federated users.	Redirect from endpoints like Authorize endpoint, `/logout`, and `/confirmforgotPassword`. See Login endpoint.
https://*Your user pool domain*/logout	Signs out user pool users.	Direct link. See Logout endpoint.
https://*Your user pool domain*/confirmUser	Confirms users who have selected an email link to verify their user account.	User selected link in an email message.
https://*Your user pool domain*/signup	Signs up a new user. The `/login` page directs your user to `/signup` when they select **Sign up**.	Direct link with same parameters as `/oauth2/authorize`.
https://*Your user pool domain*/confirm	After your user pool sends a confirmation code to a user who signed up, prompts your user for the code.	Redirect-only from `/signup`.
https://*Your user pool domain*/forgotPassword	Prompts your user for their user name and sends a password-reset code. The `/login` page directs your user to `/forgotPassword` when they select **Forgot your password?**.	1. From **Forgot password** link at `/login`. 2. Direct link with same parameters as `/oauth2/authorize`.

Endpoint URL	Description	How it's accessed
https://*Your user pool domain*/confirmforgotPassword	Prompts your user for their password-reset code and a new password. The `/forgotPassword` page directs your user to `/confirmforgotPassword` when they select **Reset your password**. After your user submits a valid reset code and new password, `/confirmforgotPassword` redirects them to `/login` to sign in with their new password.	Redirect-only from `/forgotPassword`.
https://*Your user pool domain*/resendcode	Sends a new confirmation code to a user who has signed up in your user pool.	Redirect-only from **Send a new code** link at `/confirm`.
https://*Your user pool domain*/passkeys/add	Registers a new passkey. Only available in managed login.	• In the sign-up flow after confirmation in app clients that support passkey authentication. • Direct link with same parameters as `/oauth2/authorize`.

Topics

- The managed login sign-in endpoint: /login
- The managed login sign-out endpoint: /logout

The managed login sign-in endpoint: /login

The login endpoint is an authentication server and a redirect destination from Authorize endpoint. It's the entry point to managed login when you don't specify an identity provider. When you generate a redirect to the login endpoint, it loads the login page and presents the authentication options configured for the client to the user.

> **Note**
>
> The login endpoint is a component of managed login. In your app, invoke federation and managed login pages that redirect to the login endpoint. Direct access by users to the login endpoint isn't a best practice.

GET /login

The /login endpoint only supports HTTPS GET for your user's initial request. Your app invokes the page in a browser like Chrome or Firefox. When you redirect to /login from the Authorize endpoint, it passes along all the parameters that you provided in your initial request. The login endpoint supports all the request parameters of the authorize endpoint. You can also access the login endpoint directly. As a best practice, originate all your users' sessions at /oauth2/authorize.

Example – prompt the user to sign in

This example displays the login screen.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/login?
                response_type=code&
                client_id=1example23456789&
                redirect_uri=https://YOUR_APP/redirect_uri&
                state=STATE&
                scope=openid+profile+aws.cognito.signin.user.admin
```

Example – response

The authentication server redirects to your app with the authorization code and state. The server must return the code and state in the query string parameters and not in the fragment.

```
HTTP/1.1 302 Found
                    Location: https://YOUR_APP/redirect_uri?
code=AUTHORIZATION_CODE&state=STATE
```

User-initiated sign-in request

After your user loads the `/login` endpoint, they can enter a user name and password and choose **Sign in**. When they do this, they generate an `HTTPS POST` request with the same header request parameters as the `GET` request, and a request body with their username, password, and a device fingerprint.

Step-up authentication request

The `/login` endpoint accepts the following URL parameters for step-up authentication. These parameters have the same behavior as on the authorize endpoint. For more information about ACR levels and AMR values, see the section called "Authentication levels with ACR and AMR claims".

`acr_values`

A space-separated, URL-encoded list of target ACR authentication level URIs, from highest to lowest. Amazon Cognito evaluates the list from left to right, enforces the first valid level that the user can satisfy, and ignores values that it doesn't recognize. This parameter behaves the same as the `TARGET_ACR_VALUES` API parameter. Requires the Essentials or Plus feature plan.

`max_age`

A non-negative integer that specifies the maximum number of seconds since the user last authenticated. If the current time minus `auth_time` is greater than `max_age`, the user must authenticate again from scratch instead of only stepping up. This behavior is identical to the `MAX_AGE` API parameter.

For an example request that uses these parameters, see the authorize endpoint.

Combine `acr_values` with `max_age` to require a specific authentication level with recent authentication for a sensitive operation.

The behavior on an insufficient feature plan differs from the API. The managed login `/login` endpoint silently ignores `acr_values` and proceeds with normal authentication, without an error.

The API returns a `FeatureUnavailableInTierException`. For more information, see Feature plan requirements.

The managed login sign-out endpoint: `/logout`

The `/logout` endpoint is a redirection endpoint. It signs out the user and redirects either to an authorized sign-out URL for your app client, or to the `/login` endpoint. The available parameters in a GET request to the `/logout` endpoint are tailored to Amazon Cognito managed login use cases.

The logout endpoint is a front-end web application for interactive user sessions with your customers. Your app must invoke this and other managed login endpoints in your users' browsers.

To redirect your user to managed login to sign in again, add a `redirect_uri` parameter to your request. A `logout` request with a `redirect_uri` parameter must also include parameters for your subsequent request to the Login endpoint, like `client_id`, `response_type`, and `scope`.

To redirect your user to a page that you choose, add **Allowed sign-out URLs** to your app client. In your users' requests to the `logout` endpoint, add `logout_uri` and `client_id` parameters. If the value of `logout_uri` is one of the **Allowed sign-out URLs** for your app client, Amazon Cognito redirects users to that URL.

With single logout (SLO) for SAML 2.0 IdPs, Amazon Cognito first redirects your user to the SLO endpoint you defined in your IdP configuration. After your IdP redirects your user back to `saml2/logout`, Amazon Cognito responds with one more redirect to the `redirect_uri` or `logout_uri` from your request. For more information, see Signing out SAML users with single sign-out.

The logout endpoint doesn't sign users out of OIDC or social identity providers (IdPs). To sign users out from their session with an external IdP, direct them to the sign-out page for that provider.

GET /logout

The `/logout` endpoint only supports `HTTPS GET`. The user pool client typically makes this request through the system browser. The browser is typically Custom Chrome Tab in Android or Safari View Control in iOS.

Request parameters

client_id

The app client ID for your app. To get an app client ID, you must register the app in the user pool. For more information, see Application-specific settings with app clients.

Required.

logout_uri

Redirect your user to a custom sign-out page with a *logout_uri* parameter. Set its value to the app client **sign-out URL** where you want to redirect your user after they sign out. Use *logout_uri* only with a *client_id* parameter. For more information, see Application-specific settings with app clients.

You can also use the *logout_uri* parameter to redirect your user to the sign-in page for another app client. Set the sign-in page for the other app client as an **Allowed callback URL** in your app client. In your request to the `/logout` endpoint, set the value of the *logout_uri* parameter to the URL-encoded sign-in page.

Amazon Cognito requires either a *logout_uri* or a *redirect_uri* parameter in your request to the `/logout` endpoint. A *logout_uri* parameter redirects your user to another website. If both *logout_uri* and *redirect_uri* parameters are included in your request to the `/logout` endpoint, Amazon Cognito will utilize the *logout_uri* parameter exclusively, overriding the *redirect_uri* parameter.

nonce

(Optional) A random value that you can add to the request. The nonce value that you provide is included in the ID token that Amazon Cognito issues. To guard against replay attacks, your app can inspect the `nonce` claim in the ID token and compare it to the one you generated. For more information about the `nonce` claim, see ID token validation in the *OpenID Connect standard*.

redirect_uri

Redirect your user to your sign-in page to authenticate with a *redirect_uri* parameter. Set its value to the app client **Allowed callback URL** where you want to redirect your user after they sign in again. Add *client_id*, *scope*, *state*, and *response_type* parameters that you want to pass to your `/login` endpoint.

Amazon Cognito requires either a *logout_uri* or a *redirect_uri* parameter in your request to the `/logout` endpoint. To redirect your user to your `/login` endpoint to reauthenticate

and pass tokens to your app, add a *redirect_uri* parameter. If both *logout_uri* and *redirect_uri* parameters are included in your request to the `/logout` endpoint, Amazon Cognito overrides the *redirect_uri* parameter and processes the *logout_uri* parameter exclusively.

response_type

The OAuth 2.0 response that you want to receive from Amazon Cognito after your user signs in. `code` and `token` are the valid values for the *response_type* parameter.

Required if you use a *redirect_uri* parameter.

state

When your application adds a *state* parameter to a request, Amazon Cognito returns its value to your app when the `/oauth2/logout` endpoint redirects your user.

Add this value to your requests to guard against CSRF attacks.

You can't set the value of a `state` parameter to a URL-encoded JSON string. To pass a string that matches this format in a `state` parameter, encode the string to base64, then decode it in your application.

Strongly recommended if you use a *redirect_uri* parameter.

scope

The OAuth 2.0 scopes that you want to request from Amazon Cognito after you sign them out with a *redirect_uri* parameter. Amazon Cognito redirects your user to the `/login` endpoint with the *scope* parameter in your request to the `/logout` endpoint.

Optional if you use a *redirect_uri* parameter. If you don't include a *scope* parameter, Amazon Cognito redirects your user to the `/login` endpoint with a *scope* parameter. When Amazon Cognito redirects your user and automatically populates `scope`, the parameter includes all authorized scopes for your app client.

Example requests

Example – log out and redirect user to client

Amazon Cognito redirects user sessions to the URL in the value of `logout_uri`, ignoring all other request parameters, when requests include `logout_uri` and `client_id`. This URL must be an authorized sign-out URL for the app client.

The following is an example request for sign-out and redirect to `https://www.example.com/welcome`.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?
  client_id=1example23456789&
  logout_uri=https%3A%2F%2Fwww.example.com%2Fwelcome
```

Example – log out and prompt the user to sign in as another user

When requests omit `logout_uri` but otherwise provide the parameters that make up a well-formed request to the authorize endpoint, Amazon Cognito redirects users to managed login sign-in. The logout endpoint appends the parameters in your original request to the redirect destination.

The additional parameters that you add to the logout request must be in the list at Request parameters. For example, the logout endpoint doesn't support automatic IdP redirect with `identity_provider` or `idp_identifier` parameters. The parameter `redirect_uri` in a request to the logout endpoint is not a sign-out URL, but a post-sign-in URL that you want to pass through to the authorize endpoint.

The following is an example request that signs a user out, redirects to the sign-in page, and provides an authorization code to `https://www.example.com` after sign-in.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/logout?
  response_type=code&
  client_id=1example23456789&
  redirect_uri=https%3A%2F%2Fwww.example.com&
  state=example-state-value&
  nonce=example-nonce-value&
  scope=openid+profile+aws.cognito.signin.user.admin
```

Identity provider and relying party endpoints

Federation endpoints are user pool endpoints that serve a purpose for one of the authentication standards used by user pools. They include SAML ACS URLs, OIDC discovery endpoints, and service endpoints for user pool roles both as identity provider and relying party. Federation endpoints initiate authentication flows, receive proof of authentication from IdPs, and issue tokens to clients. They interact with IdPs, applications, and administrators, but not with users.

The full-page topics after this page have details about the OAuth 2.0 and OIDC provider endpoints that become available when you add a domain to your user pool. The following chart is a list of all federation endpoints.

Examples of user pool domains are:

1. Prefix domain: `mydomain.auth.us-east-1.amazoncognito.com`
2. Custom domain: `auth.example.com`

User pool federation endpoints

Endpoint URL	Description	How it's accessed
https://*Your user pool domain*/oauth2/authorize	Redirects a user to either managed login or to sign in with their IdP.	Invoked in customer browser to begin user authentication. See Authorize endpoint.
https://*Your user pool domain*/oauth2/token	Returns tokens based on an authorization code or client credentials request.	Requested by app to retrieve tokens. See Token endpoint.
https://*Your user pool domain*/oauth2/userInfo	Returns user attributes based on OAuth 2.0 scopes and user identity in an access token.	Requested by app to retrieve user profile. See userInfo endpoint.
https://*Your user pool domain*/oauth2/revoke	Revokes a refresh token and the associated access tokens.	Requested by app to revoke a token. See Revoke endpoint.
https://cognito-idp.*Region*.amazonaws.com/*your user pool ID*/.well-known/openid-configuration	A directory of the OIDC architecture of your user pool.[1]	Requested by app to locate user pool issuer metadata.
https://cognito-idp.*Region*.amazonaws.com/*your user pool ID*/.well-known/jwks.json	Public keys that you can use to validate Amazon Cognito tokens.[2]	Requested by app to verify JWTs.

Endpoint URL	Description	How it's accessed
https://*Your user pool domain*/oauth2/idpresponse	Social identity providers must redirect your users to this endpoint with an authorization code. Amazon Cognito redeems the code for a token when it authenticates your federated user.	Redirected from OIDC IdP sign-in as the IdP client callback URL.
https://*Your user pool domain*/saml2/idpresponse	The Assertion Consumer Response (ACS) URL for integration with SAML 2.0 identity providers.	Redirected from SAML 2.0 IdP as the ACS URL, or the origination point for IdP-initiated sign-in[3].
https://*Your user pool domain*/saml2/logout	The Single Logout (SLO) URL for integration with SAML 2.0 identity providers.	Redirected from SAML 2.0 IdP as the single logout (SLO) URL. Accepts POST binding only.

[1] The `openid-configuration` document might be updated at any time with additional information that keeps the endpoint compliant with the OIDC and OAuth2 specifications.

[2]The `jwks.json` JSON file might be updated at any time to with new public token signing keys.

[3] For more information about IdP-initiated SAML sign-in, see Implement IdP-initiated SAML sign-in.

For more information on the OpenID Connect and OAuth standards, see OpenID Connect 1.0 and OAuth 2.0.

Amazon Cognito user pools as an OIDC issuer

Amazon Cognito user pools function as OpenID Connect (OIDC) identity providers, serving as an issuer that application libraries can use for OIDC federation. Application libraries for OIDC federation can reference the two different paths as discussed below as an autodiscovery endpoint. This endpoint provides access to the JSON Web Key Set (JWKS) at `/.well-known/jwks.json` and OIDC discovery metadata at `/.well-known/openid-configuration`, where applications can discover the authorization, token, and userInfo endpoints.

Applications that support OIDC autodiscovery can automatically configure themselves by querying these well-known endpoints. For applications that don't support autodiscovery, you can hardcode your application with the specific OIDC endpoints that are listed in the previous section.

User pool issuer types

Original issuer

The current default issuer configuration for user pools. The issuer URL is hosted in the user pool's Region and provides OIDC endpoints specific to that Region.

Original issuers take the format `https://cognito-idp.`*`eusc-de-east-1`*`.amazonaws.com/`*`eusc-de-east-1_EXAMPLE`*.

Updated issuer

Recommended for all user pools, including for multi-Region replication. Updated issuers host the same JWKS content in multiple Regions, resulting in improved resilience and efficiency.

Updated issuers take the format `https://issuer-cognito-idp.`*`eusc-de-east-1`*`.amazonaws.com/`*`eusc-de-east-1_EXAMPLE`*, where *`Region`* is the primary AWS Region of your user pool.

> **Note**
>
> The updated issuer type is not currently compatible with Application Load Balancer authentication with Amazon Cognito (Authenticate users using an Application Load Balancer) or Amazon API Gateway Amazon Cognito authorizers (Control access to REST APIs using Amazon Cognito user pools as authorizer).

Prepare for changes to the discovery document

The `openid-configuration` document is a standard OpenID Connect discovery document. Amazon Cognito can add metadata entries to it over time to keep it compliant with the OpenID Connect and OAuth 2.0 specifications and to improve compatibility with OIDC client libraries. These additions are backward compatible: Amazon Cognito adds new members, but it does not rename, change the type of, or remove the members that your application already reads. Write your client so that it keeps working as the document grows.

Ignore members that you don't recognize

As the OpenID Connect Discovery specification requires, treat metadata members that you don't understand as optional. Don't reject, fail, or throw an error when the document contains a member that your application doesn't use.

Don't hard-code the set or order of members

Parse the document as a general JSON object rather than a fixed structure. The members can appear in any order, and the set of members can change between requests and over time. Address each member by name, and read each one as the type that the specification defines for it.

Test strict clients against the current document

If your client validates the discovery document strictly, fetch the current document from your user pool's `/.well-known/openid-configuration` endpoint and confirm that your client accepts it before you depend on a specific set of members.

Amazon Cognito tokens follow the same additive, forward-compatible approach. For more information about parsing the claims in an ID or access token, see Parse user pool tokens safely.

Topics

- The redirect and authorization endpoint
- The token endpoint
- The user attributes endpoint
- The token revocation endpoint
- The IdP SAML assertion endpoint

The redirect and authorization endpoint

The `/oauth2/authorize` endpoint is a redirection endpoint that supports two redirect destinations. If you include an `identity_provider` or `idp_identifier` parameter in the URL, it silently redirects your user to the sign-in page for that identity provider (IdP). Otherwise, it redirects to the Login endpoint with the same URL parameters that you included in your request.

The authorize endpoint redirects either to managed login or to an IdP sign-in page. The destination of a user session at this endpoint is a webpage that your user must interact with directly in their browser.

To use the authorize endpoint, invoke your user's browser at `/oauth2/authorize` with parameters that provide your user pool with information about the following user pool details.

- The app client that you want to sign in to.
- The callback URL that you want to end up at.
- The OAuth 2.0 scopes that you want to request in your user's access token.
- Optionally, the third-party IdP that you want to use to sign in.

You can also supply `state` and `nonce` parameters that Amazon Cognito uses to validate incoming claims.

GET /oauth2/authorize

The `/oauth2/authorize` endpoint only supports `HTTPS GET`. Your app typically initiates this request in your user's browser. You can only make requests to the `/oauth2/authorize` endpoint over HTTPS.

You can learn more about the definition of the authorization endpoint in the OpenID Connect (OIDC) standard at Authorization Endpoint.

Request parameters

`response_type`

Required.

The response type. Must be `code` or `token`.

A successful request with a `response_type` of `code` returns an authorization code grant. An authorization code grant is a `code` parameter that Amazon Cognito appends to your redirect URL. Your app can exchange the code with the Token endpoint for access, ID, and refresh tokens. As a security best practice, and to receive refresh tokens for your users, use an authorization code grant in your app.

A successful request with a `response_type` of `token` returns an implicit grant. An implicit grant is an ID and access token that Amazon Cognito appends to your redirect URL. An implicit grant is less secure because it exposes tokens and potential identifying information to users. You can deactivate support for implicit grants in the configuration of your app client.

`client_id`

Required.

The app client ID.

The value of `client_id` must be the ID of an app client in the user pool where you make the request. Your app client must support sign-in by Amazon Cognito local users or at least one third-party IdP.

`redirect_uri`

Required.

The URL where the authentication server redirects the browser after Amazon Cognito authorizes the user.

A redirect uniform resource identifier (URI) must have the following attributes:

- It must be an absolute URI.
- You must have pre-registered the URI with a client.
- It can't include a fragment component.

See OAuth 2.0 - Redirection Endpoint.

Amazon Cognito requires that your redirect URI use HTTPS, except for `http://localhost`, which you can set as a callback URL for testing purposes.

Amazon Cognito also supports app callback URLs such as `myapp://example`.

`state`

Optional, recommended.

When your app adds a *state* parameter to a request, Amazon Cognito returns its value to your app when the `/oauth2/authorize` endpoint redirects your user.

Add this value to your requests to guard against CSRF attacks.

You can't set the value of a `state` parameter to a URL-encoded JSON string. To pass a string that matches this format in a `state` parameter, encode the string to base64, then decode it in your app.

`identity_provider`

Optional.

Add this parameter to bypass managed login and redirect your user to a provider sign-in page. The value of the *identity_provider* parameter is the name of the identity provider (IdP) as it appears in your user pool.

- For social providers, you can use the *identity_provider* values `Facebook`, `Google`, `LoginWithAmazon`, and `SignInWithApple`.
- For Amazon Cognito user pools, use the value `COGNITO`.
- For SAML 2.0 and OpenID Connect (OIDC) identity providers (IdPs), use the name that you assigned to the IdP in your user pool.

`idp_identifier`

Optional.

Add this parameter to redirect to a provider with an alternative name for the *identity_provider* name. You can enter identifiers for your SAML 2.0 and OIDC IdPs from the **Social and external providers** menu of the Amazon Cognito console.

`scope`

Optional.

Can be a combination of any system-reserved scopes or custom scopes that are associated with a client. Scopes must be separated by spaces. System reserved scopes are `openid`, `email`, phone, `profile`, and `aws.cognito.signin.user.admin`. Any scope used must be associated with the client, or it will be ignored at runtime.

If the client doesn't request any scopes, the authentication server uses all scopes that are associated with the client.

An ID token is only returned if `openid` scope is requested. The access token can be only used against Amazon Cognito user pools if `aws.cognito.signin.user.admin` scope is requested. The phone, `email`, and `profile` scopes can only be requested if `openid` scope is also requested. These scopes dictate the claims that go inside the ID token.

`code_challenge_method`

Optional.

The hashing protocol that you used to generate the challenge. The PKCE RFC defines two methods, S256 and plain; however, Amazon Cognito authentication server supports only S256.

`code_challenge`

Optional.

The proof of key code exchange (PKCE) challenge that you generated from the `code_verifier`. For more information, see Using PKCE in authorization code grants.

Required only when you specify a `code_challenge_method` parameter.

`nonce`

Optional.

A random value that you can add to the request. The nonce value that you provide is included in the ID token that Amazon Cognito issues. To guard against replay attacks, your app can inspect the `nonce` claim in the ID token and compare it to the one you generated. For more information about the `nonce` claim, see ID token validation in the *OpenID Connect standard*.

`lang`

Optional.

The language that you want to display user-interactive pages in. Managed login pages can be localized, but hosted UI (classic) pages can't. For more information, see Managed login localization.

`login_hint`

Optional.

A username prompt that you want to pass to the authorization server. You can collect a username, email address or phone number from your user and allow the destination provider to pre-populate the user's sign-in name. When you submit a `login_hint` parameter and no `idp_identifier` or `identity_provider` parameters to the `oauth2/authorize` endpoint, managed login fills the username field with your hint value. You can also pass this parameter to the Login endpoint and automatically fill the username value.

When your authorization request invokes a redirect to OIDC IdPs, Amazon Cognito adds a `login_hint` parameter to the request to that third-party authorizer. You can't forward login hints to SAML, Apple, Login With Amazon, Google, or Facebook (Meta) IdPs.

`prompt`

Optional.

An OIDC parameter that controls authentication behavior for existing sessions. Available in the managed login branding version only, not in the classic hosted UI. For more information from the OIDC specification, see Authentication request. The values `none` and `login` have an effect on user pool authentication behavior.

Amazon Cognito forwards all values of `prompt` except `none` to your IdPs when users select authentication with third-party providers. This is true when the URL that users access includes an `identity_provider` or `idp_identifier` parameter, or when the authorization server redirects them to the Login endpoint and they select and IdP from the available buttons.

Prompt parameter values

`prompt=none`

Amazon Cognito silently continues authentication for users who have a valid authenticated session. With this prompt, users can silently authenticate between different app clients in your user pool. If the user is not already authenticated, the authorization server returns a `login_required` error.

`prompt=login`

Amazon Cognito requires users to re-authenticate even if they have an existing session. Send this value when you want to verify the user's identity again. Authenticated users who have an existing session can return to sign-in without invalidating that session. When a user who has an existing session signs in again, Amazon Cognito assigns them a new session cookie. This parameter can also be forwarded to your IdPs. IdPs that accept this parameter also request a new authentication attempt from the user.

`prompt=select_account`

This value has no effect on local sign-in and must be submitted in requests that redirect to IdPs. When included in your authorization request, this parameter adds `prompt=select_account` to the URL path for the IdP redirect destination. When IdPs support this parameter, they request that users select the account that they want to log in with.

`prompt=consent`

This value has no effect on local sign-in and must be submitted in requests that redirect to IdPs. When included in your authorization request, this parameter adds `prompt=consent` to the URL path for the IdP redirect destination. When IdPs support this parameter, they request user consent before they redirect back to your user pool.

When you omit the `prompt` parameter from your request, managed login follows the default behavior: users must sign in unless their browser has a valid managed login session cookie. You can combine multiple values for `prompt` with a space-character delimiter, for example `prompt=login consent`.

`resource`

Optional.

The identifier of a resource that you want to bind to the access token in the aud claim. When you include this parameter, Amazon Cognito validates that the value is a URL and sets the audience of the resulting access token to the requested resource. You can request a user pool resource server with an identifier in a URL format, or a URL of your choosing. Values for this parameter must begin with `https://`, `http://localhost`, or a custom URL scheme like `myapp://`.

Resource binding is defined in RFC 8707. For more information about resource servers and resource binding, see Resource binding.

Example: authorization code grant

This is an example request for an authorization code grant.

The following request initiates a session to retrieve an authorization code that your user passes to your app at the `redirect_uri` destination. This session requests scopes for user attributes and for access to Amazon Cognito self-service API operations.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin
```

The Amazon Cognito authentication server redirects back to your app with the authorization code and state. The authorization code is valid for five minutes.

```
HTTP/1.1 302 Found
Location: https://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

Example: authorization code grant with PKCE

This example flow performs an authorization code grant with PKCE.

This request adds a `code_challenge` parameter. To complete the exchange of a code for a token, you must include the `code_verifier` parameter in your request to the `/oauth2/token` endpoint.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=aws.cognito.signin.user.admin&
code_challenge_method=S256&
code_challenge=a1b2c3d4...
```

The authorization server redirects back to your application with the authorization code and state. Your application processes the authorization code and exchanges it for tokens.

```
HTTP/1.1 302 Found
Location: https://www.example.com?code=a1b2c3d4-5678-90ab-cdef-
EXAMPLE11111&state=abcdefg
```

Example: require re-authentication with `prompt=login`

The following request adds a `prompt=login` parameter that requires the user to authenticate again, even if they have an existing session.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin&
prompt=login
```

The authorization server redirects to the login endpoint, requiring re-authentication.

```
HTTP/1.1 302 Found Location: https://mydomain.auth.us-east-1.amazoncognito.com/
login?response_type=code&client_id=1example23456789&redirect_uri=https://
```

```
www.example.com&state=abcdefg&scope=openid+profile
+aws.cognito.signin.user.admin&prompt=login
```

Example: silent authentication with `prompt=none`

The following request adds a `prompt=none` parameter that silently checks if the user has a valid session.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=openid+profile+aws.cognito.signin.user.admin&
prompt=none
```

When no valid session exists, the authorization server returns an error to the redirect URI

```
HTTP/1.1 302 Found Location: https://www.example.com?error=login_required&state=abcdefg
```

When a valid session exists, the authorization server returns an authorization code.

```
HTTP/1.1 302 Found Location: https://www.example.com?
code=AUTHORIZATION_CODE&state=abcdefg
```

Example: authorization code grant with resource binding

The following request adds a `resource` parameter to bind the access token to a specific resource server. The resulting access token creates the conditions for the target API to validate that it is the intended audience of the authenticated user's request.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=solar-system-data-api.example.com/asteroids.add&
resource=https://solar-system-data-api.example.com
```

The authorization server returns an authorization code that results in an access token with an `aud` claim of `https://solar-system-data-api.example.com`.

```
HTTP/1.1 302 Found Location: https://www.example.com?
code=AUTHORIZATION_CODE&state=abcdefg
```

Example: Token (implicit) grant without openid scope

This example flow generates an implicit grant and returns JWTs directly to the user's session.

The request is for an implicit grant from your authorization server. It requests scopes in the access token that authorize user profile self-service operations.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=token&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=aws.cognito.signin.user.admin
```

The authorization server redirects back to your application with an access token only. Because openid scope was not requested, Amazon Cognito doesn't return an ID token. Also, Amazon Cognito doesn't return a refresh token in this flow.

```
HTTP/1.1 302 Found
Location: https://example.com/
callback#access_token=eyJra456defEXAMPLE&token_type=bearer&expires_in=3600&state=STATE
```

Example: Token (implicit) grant with openid scope

This example flow generates an implicit grant and returns tokens to the user's browser.

The request is for an implicit grant from your authorization server. It requests scopes in the access token that authorize access to user attributes and self-service operations.

```
GET
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=token&
client_id=1example23456789&
redirect_uri=https://www.example.com&
state=abcdefg&
scope=aws.cognito.signin.user.admin+openid+profile
```

The authorization server redirects back to your application with access token and ID token (because `openid` scope was included):

```
HTTP/1.1 302 Found
Location: https://
www.example.com#id_token=eyJra67890EXAMPLE&access_token=eyJra12345EXAMPLE&token_type=bearer&exp
```

Examples of negative responses

Amazon Cognito might deny your request. Negative requests come with an HTTP error code and a description that you can use to correct your request parameters. The following are examples of negative responses.

- If `client_id` and `redirect_uri` are valid, but the request parameters aren't formatted correctly, the authentication server redirects the error to the client's `redirect_uri` and appends an error message in a URL parameter. The following are examples of incorrect formatting.
  - The request doesn't include a `response_type` parameter.
  - The authorization request provided a `code_challenge` parameter, but not a `code_challenge_method` parameter.
  - The value of the `code_challenge_method` parameter isn't `S256`.

  The following is the response to an example request with incorrect formatting.

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_request
  ```

- If the client requests `code` or `token` in `response_type`, but doesn't have permission for these requests, the Amazon Cognito authorization server returns `unauthorized_client` to client's `redirect_uri`, as follows:

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=unauthorized_client
  ```

- If the client requests scope that is unknown, malformed, or not valid, the Amazon Cognito authorization server returns `invalid_scope` to the client's `redirect_uri`, as follows:

  ```
  HTTP 1.1 302 Found Location: https://client_redirect_uri?error=invalid_scope
  ```

- If there is any unexpected error in the server, the authentication server returns `server_error` to the client's `redirect_uri`. Because the HTTP 500 error doesn't get sent to the client, the error doesn't display in the user's browser. The authorization server returns the following error.

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?error=server_error
```

- When Amazon Cognito authenticates through federation to third-party IdPs, Amazon Cognito might experience connection issues, such as the following:
  - If a connection timeout occurs while requesting token from the IdP, the authentication server redirects the error to the client's `redirect_uri` as follows:

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=Timeout+occurred+in+calling+IdP+token
+endpoint
```

  - If a connection timeout occurs while calling the `jwks_uri` endpoint for ID token validation, the authentication server redirects with an error to the client's `redirect_uri` as follows:

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=error_description=Timeout+in+calling+jwks
+uri
```

- When authenticating by federating to third-party IdPs, the providers may return error responses. This can be due to configuration errors or other reasons, such as the following:
  - If an error response is received from other providers, the authentication server redirects the error to the client's `redirect_uri` as follows:

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=[IdP name]+Error+-+[status code]+error
 getting token
```

  - If an error response is received from Google, the authentication server redirects the error to the client's `redirect_uri` as follows:

```
HTTP 1.1 302 Found Location: https://client_redirect_uri?
error=invalid_request&error_description=Google+Error+-+[status code]+[Google-
provided error code]
```

- When Amazon Cognito encounters an communication exception when it connects to an external IdP, the authentication server redirects with an error to the client's `redirect_uri` with either of the following messages:
  - ```
    HTTP 1.1 302 Found Location: https://client_redirect_uri?
    error=invalid_request&error_description=Connection+reset
    ```
  - ```
    HTTP 1.1 302 Found Location: https://client_redirect_uri?
    error=invalid_request&error_description=Read+timed+out
    ```

Example: Step-up authentication request

Amazon Cognito supports step-up authentication with managed login. It isn't supported in the hosted UI. If your application relies on ACR and AMR, use managed login.

For more information about ACR levels and AMR values, see the section called “Authentication levels with ACR and AMR claims”.

The `/oauth2/authorize` endpoint accepts the following URL parameters for step-up authentication.

`acr_values`

A space-separated, URL-encoded list of target ACR authentication level URIs, from highest to lowest. Amazon Cognito evaluates the list from left to right, enforces the first valid level that the user can satisfy, and ignores values that it doesn't recognize. This parameter behaves the same as the `TARGET_ACR_VALUES` API parameter. Requires the Essentials or Plus feature plan.

`max_age`

A non-negative integer that specifies the maximum number of seconds since the user last authenticated. If the current time minus `auth_time` is greater than `max_age`, the user must authenticate again from scratch instead of only stepping up. This behavior is identical to the `MAX_AGE` API parameter.

Combine `acr_values` with `max_age` to require a specific authentication level with recent authentication for a sensitive operation.

The following example `/oauth2/authorize` request asks Amazon Cognito to try level 4 first. If the user can't satisfy level 4, for example because they have no TOTP configured, Amazon Cognito

falls back to level 3, assuming the user has an SMS or email one-time password configured. The request also requires that the user authenticated within the last 3,600 seconds. The example client ID *1example23456789* and the other italicized values are placeholders. Replace them with your own values.

```
GET https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://YOUR_APP/
redirect_uri&state=STATE&scope=openid+profile
+aws.cognito.signin.user.admin&acr_values=urn%3Acognito%3Aloa%3A4+urn%3Acognito%3Aloa
%3A3&max_age=3600
```

The behavior on an insufficient feature plan differs from the API. The managed login authorize endpoint silently ignores `acr_values` and proceeds with normal authentication, without an error. The API returns a `FeatureUnavailableInTierException`. For more information, see Feature plan requirements.

The token endpoint

The OAuth 2.0 token endpoint at `/oauth2/token` issues JSON web tokens (JWTs) to applications that want to complete authorization-code and client-credentials grant flows. These tokens are the end result of authentication with a user pool. They contain information about the user (ID token), the user's level of access (access token), and the user's entitlement to persist their signed-in session (refresh token). OpenID Connect (OIDC) relying-party libraries handle requests to and response payloads from this endpoint. Tokens provide verifiable proof of authentication, profile information, and a mechanism for access to back-end systems.

The token endpoint returns an `Access-Control-Allow-Origin: *` response header. You can call the token endpoint cross-origin from a browser-based application. For example, you can complete an authorization code grant with Proof Key for Code Exchange (PKCE) from a public client. Amazon Cognito does not support custom cross-origin resource sharing (CORS) origin policies on this endpoint. For more information, see the CORS policies section of User pool managed login.

Your user pool OAuth 2.0 authorization server issues JSON web tokens (JWTs) from the token endpoint to the following types of sessions:

1. Users who have completed a request for an authorization code grant. Successful redemption of a code returns ID, access, and refresh tokens.

2. Machine-to-machine (M2M) sessions that have completed a client-credentials grant. Successful authorization with the client secret returns an access token.
3. Users who have previously signed in and received refresh tokens. Refresh token authentication returns new ID and access tokens.

> **Note**
>
> Users who sign in with an authorization code grant in managed login or through federation can always refresh their tokens from the token endpoint. Users who sign in with the API operations `InitiateAuth` and `AdminInitiateAuth` can refresh their tokens with the token endpoint when remembered devices is *not* active in your user pool. If remembered devices is active, refresh tokens with the relevant API or SDK token-refresh operation for your app client.

The token endpoint becomes publicly available when you add a domain to your user pool. It accepts HTTP POST requests. For application security, use PKCE with your authorization code sign-in events. PKCE verifies that the user passing an authorization code is that same user who authenticated. For more information about PKCE, see IETF RFC 7636.

You can learn more about the user pool app clients and their grant types, client secrets, allowed scopes, and client IDs at Application-specific settings with app clients. You can learn more about M2M authorization, client credentials grants, and authorization with access token scopes at Scopes, M2M, and resource servers.

To retrieve information about a user from their access token, pass it to your userInfo endpoint or to a GetUser API request. The access token must contain the appropriate scopes for these requests,

How Amazon Cognito differs from the OIDC specification

Amazon Cognito doesn't implement the OpenID Connect `offline_access` scope model for controlling refresh-token issuance. Instead, the token endpoint returns a refresh token whenever the `grant_type` is `authorization_code`, and you control refresh-token lifetime through the app client's refresh-token validity setting rather than by requesting `offline_access`. Requesting the `offline_access` scope has no effect, and omitting it does not suppress the refresh token.

Format a POST request to the token endpoint

The `/oauth2/token` endpoint only supports `HTTPS POST`. This endpoint is not user-interactive. Handle token requests with an OpenID Connect (OIDC) library in your application.

The token endpoint supports `client_secret_basic` and `client_secret_post` authentication. For more information about the OIDC specification, see Client Authentication. For more information about the token endpoint from the OpenID Connect specification, see Token Endpoint.

Request parameters in header

You can pass the following parameters in the header of your request to the token endpoint.

`Authorization`

If the client was issued a secret, the client can pass its `client_id` and `client_secret` in the authorization header as `client_secret_basic` HTTP authorization. You can also include the `client_id` and `client_secret` in the request body as `client_secret_post` authorization.

The authorization header string is Basic `Base64Encode(client_id:client_secret)`. The following example is an authorization header for app client `djc98u3jiedmi283eu928` with client secret `abcdef01234567890`, using the Base64-encoded version of the string `djc98u3jiedmi283eu928:abcdef01234567890`:

```
Authorization: Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw
```

`Content-Type`

Set the value of this parameter to `'application/x-www-form-urlencoded'`.

Request parameters in body

The following are parameters that you can request in `x-www-form-urlencoded` format in the request body to the token endpoint.

`grant_type`

Required.

The type of OIDC grant that you want to request.

Must be `authorization_code` or `refresh_token` or `client_credentials`. You can request an access token for a custom scope from the token endpoint under the following conditions:

- You enabled the requested scope in your app client configuration.
- You configured your app client with a client secret.
- You enabled the app client for the client credentials grant. You can enable this grant in one of the following ways:
  - For hosted-domain machine-to-machine (M2M) authentication, add `client_credentials` to the app client's `AllowedOAuthFlows`.
  - For domainless M2M authentication, add `ALLOW_CLIENT_TOKEN_AUTH` to the app client's `ExplicitAuthFlows`. This flow requires a client secret and must be the app client's only authentication flow.

  If you enable neither of these options, the token endpoint returns an unauthorized-class error.

> **Note**
>
> The token endpoint returns a refresh token only when the `grant_type` is `authorization_code`.

`client_id`

Optional. Not required when you provide the app client ID in the `Authorization` header.

The ID of an app client in your user pool. Specify the same app client that authenticated your user.

You must provide this parameter if the client is public and does not have a secret, or with `client_secret` in `client_secret_post` authorization.

`client_secret`

Optional. Not required when you provide the client secret in the `Authorization` header and when the app client doesn't have a secret.

The app client secret, if the app client has one, for `client_secret_post` authorization.

`scope`

Optional.

Can be a combination of any scopes that are associated with your app client. If the request includes a scope that is defined on a resource server, Amazon Cognito checks whether that scope is allowed for the requested app client. If the scope isn't allowed, Amazon Cognito rejects the entire request rather than issuing a token for the remaining scopes. A request that combines allowed and unallowed scopes fails for the same reason.

Amazon Cognito rejects an unusable scope rather than dropping it and returning a token for the rest. A rejected request returns `HTTP 400` with an error in the response body. The error value depends on why the scope was unusable. When a scope exists but isn't allowed for the app client, Amazon Cognito returns `invalid_grant`. When a scope refers to a resource server or custom scope that doesn't exist, Amazon Cognito returns `invalid_scope`. If you don't provide this request parameter, the authorization server returns an access token `scope` claim with all authorization scopes that you enabled in your app client configuration. You can request any of the scopes allowed for the requested app client: standard scopes, custom scopes from resource servers, and the `aws.cognito.signin.user.admin` user self-service scope.

`redirect_uri`

Optional. Not required for client-credentials grants.

Must be the same `redirect_uri` that was used to get `authorization_code` in `/oauth2/authorize`.

You must provide this parameter if `grant_type` is `authorization_code`.

`refresh_token`

Optional. Used only when the user already has a refresh token and wishes to get new ID and access tokens.

To generate new access and ID tokens for a user's session, set the value of `refresh_token` to a valid refresh token that the requested app client issued.

Returns a new refresh token with new ID and access token when refresh token rotation is active, otherwise returns only ID and access tokens. If the original access token was bound to an API resource, the new access token maintains the requested API url in the aud claim.

code

Optional. Only required in authorization-code grants.

The authorization code from an authorization code grant. You must provide this parameter if your authorization request included a `grant_type` of `authorization_code`.

aws_client_metadata

Optional.

Information that you want to pass to the Pre token generation Lambda trigger in machine-to-machine (M2M) authorization flows. Your application can collect context information about the session and pass it in this parameter. When you pass `aws_client_metadata` in URL-encoded JSON format, Amazon Cognito includes it in the input event to your trigger Lambda function. Your pre token trigger event version or global Lambda trigger version must be configured for version three or later. Although Amazon Cognito accepts requests to this endpoint in authorization code and client credentials M2M flows, your user pool only passes `aws_client_metadata` to the pre token generation trigger from client credentials requests.

code_verifier

Optional. Required only if you provided `code_challenge_method` and `code_challenge` parameters in your initial authorization request.

The generated code verifier that your application calculated the `code_challenge` from in an authorization code grant request with PKCE.

Exchanging an authorization code for tokens

The following request successfully generates ID, access, and refresh tokens after authentication with an authorization-code grant. The request passes the client secret in `client_secret_basic` format in the `Authorization` header.

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token&
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=authorization_code&
client_id=1example23456789&
code=AUTHORIZATION_CODE&
redirect_uri=com.myclientapp://myclient/redirect
```

The response issues new ID, access, and refresh tokens to the user, with additional metadata.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "refresh_token": "eyJj3example",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Client credentials with basic authorization

The following request from an M2M application requests a client credentials grant. Because client credentials requires a client secret, the request is authorized with an `Authorization` header derived from the app client ID and secret. The request results in an access token with the two requested scopes. The request also includes client metadata that provides IP-address information and a token issued to the user who this grant is on behalf of. Amazon Cognito passes the client metadata to the pre token generation Lambda trigger.

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=client_credentials&
client_id=1example23456789&
scope=resourceServerIdentifier1%2Fscope1%20resourceServerIdentifier2%2Fscope2&
&aws_client_metadata=%7B%22onBehalfOfToken%22%3A%22eyJra789ghiEXAMPLE%22,
%20%22ClientIpAddress%22%3A%22192.0.2.252%22%7D
```

Amazon Cognito passes the following input event to the pre token generation Lambda trigger.

```
{
    version: '3',
    triggerSource: 'TokenGeneration_ClientCredentials',
    region: 'us-east-1',
    userPoolId: 'us-east-1_EXAMPLE',
    userName: 'ClientCredentials',
    callerContext: {
```

```
        awsSdkVersion: 'aws-sdk-unknown-unknown',
        clientId: '1example23456789'
    },
    request: {
        userAttributes: {},
        groupConfiguration: null,
        scopes: [
           'resourceServerIdentifier1/scope1',
           'resourceServerIdentifier2/scope2'
        ],
        clientMetadata: {
            'onBehalfOfToken': 'eyJra789ghiEXAMPLE',
            'ClientIpAddress': '192.0.2.252'
        }
    },
    response: { claimsAndScopeOverrideDetails: null }
}
```

The response returns an access token. Client credentials grants are for machine-to-machine (M2M) authorization and only return access tokens.

```
HTTP/1.1 200 OK
Content-Type: application/json
{
    "access_token": "eyJra1example",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Client credentials with POST body authorization

The following client-credentials grant request includes the `client_secret` parameter in the request body and doesn't include an `Authorization` header. This request uses the `client_secret_post` authorization syntax. The request results in an access token with the requested scope. The request also includes client metadata that provides IP-address information and a token issued to the user who this grant is on behalf of. Amazon Cognito passes the client metadata to the pre token generation Lambda trigger.

```
POST /oauth2/token HTTP/1.1
Content-Type: application/x-www-form-urlencoded
X-Amz-Target: AWSCognitoIdentityProviderService.Client credentials request
```

```
User-Agent: USER_AGENT
Accept: /
Accept-Encoding: gzip, deflate, br
Content-Length: 177
Referer: http://auth.example.com/oauth2/token
Host: auth.example.com
Connection: keep-alive

grant_type=client_credentials&
client_id=1example23456789&
scope=my_resource_server_identifier%2Fmy_custom_scope&
client_secret=9example87654321&
aws_client_metadata=%7B%22onBehalfOfToken%22%3A%22eyJra789ghiEXAMPLE%22,
%20%22ClientIpAddress%22%3A%22192.0.2.252%22%7D
```

Amazon Cognito passes the following input event to the pre token generation Lambda trigger.

```
{
    version: '3',
    triggerSource: 'TokenGeneration_ClientCredentials',
    region: 'us-east-1',
    userPoolId: 'us-east-1_EXAMPLE',
    userName: 'ClientCredentials',
    callerContext: {
        awsSdkVersion: 'aws-sdk-unknown-unknown',
        clientId: '1example23456789'
    },
    request: {
        userAttributes: {},
        groupConfiguration: null,
        scopes: [
           'resourceServerIdentifier1/my_custom_scope'
        ],
        clientMetadata: {
            'onBehalfOfToken': 'eyJra789ghiEXAMPLE',
            'ClientIpAddress': '192.0.2.252'
        }
    },
    response: { claimsAndScopeOverrideDetails: null }
}
```

The response returns an access token. Client credentials grants are for machine-to-machine (M2M) authorization and only return access tokens.

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Date: Tue, 05 Dec 2023 16:11:11 GMT
x-amz-cognito-request-id: 829f4fe2-a1ee-476e-b834-5cd85c03373b

{
    "access_token": "eyJra12345EXAMPLE",
    "expires_in": 3600,
    "token_type": "Bearer"
}
```

Authorization code grant with PKCE

The following example request completes an authorization request that included `code_challenge_method` and `code_challenge` parameters in an authorization code grant request with PKCE.

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=authorization_code&
client_id=1example23456789&
code=AUTHORIZATION_CODE&
code_verifier=CODE_VERIFIER&
redirect_uri=com.myclientapp://myclient/redirect
```

The response returns ID, access, and refresh tokens from the successful PKCE verification by the application.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "refresh_token": "eyJj3example",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Token refresh without refresh token rotation

The following example requests provides a refresh token to an app client where refresh token rotation is inactive. Because the app client has a client secret, the request provides an `Authorization` header.

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=refresh_token&
client_id=1example23456789&
refresh_token=eyJj3example
```

The response returns new ID and access tokens.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Token refresh with refresh token rotation

The following example requests provides a refresh token to an app client where refresh token rotation is active. Because the app client has a client secret, the request provides an `Authorization` header.

```
POST https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/token >
Content-Type='application/x-www-form-urlencoded'&
Authorization=Basic ZGpjOTh1M2ppZWRtaTI4M2V1OTI4OmFiY2RlZjAxMjM0NTY3ODkw

grant_type=refresh_token&
client_id=1example23456789&
refresh_token=eyJj3example
```

The response returns new ID, access, and refresh tokens.

```
HTTP/1.1 200 OK
Content-Type: application/json

{
    "access_token": "eyJra1example",
    "id_token": "eyJra2example",
    "refresh_token": "eyJj4example",
    "token_type": "Bearer",
    "expires_in": 3600
}
```

Examples of negative responses

Malformed requests generate errors from the token endpoint. The following is a general map of the response body when token requests generate an error.

```
HTTP/1.1 400 Bad Request
Content-Type: application/json;charset=UTF-8

{
"error":"invalid_request|invalid_client|invalid_grant|unauthorized_client|
unsupported_grant_type"
}
```

`invalid_request`

The request is missing a required parameter, includes an unsupported parameter value (other than `unsupported_grant_type`), or is otherwise malformed. For example, `grant_type` is `refresh_token` but `refresh_token` is not included.

`invalid_client`

Client authentication failed. For example, when the client includes `client_id` and `client_secret` in the authorization header, but there's no such client with that `client_id` and `client_secret`.

`invalid_grant`

Refresh token has been revoked.

Authorization code has been consumed already or does not exist.

App client doesn't have read access to all attributes in the requested scope. For example, your app requests the `email` scope and your app client can read the `email` attribute, but not `email_verified`.

`unauthorized_client`

Client is not allowed for code grant flow or for refreshing tokens.

A token request with a `redirect_uri` that does not match the value from the authorization request, returns `unauthorized_client` with an `error_description` of `invalid_redirect`.

`unsupported_grant_type`

Returned if `grant_type` is anything other than `authorization_code` or `refresh_token` or `client_credentials`.

The user attributes endpoint

Where OIDC issues ID tokens that contain user attributes, OAuth 2.0 implements the `/oauth2/userInfo` endpoint. An authenticated user or client receives an access token with a `scopes` claim. This claim determines the attributes that the authorization server should return. When an application presents an access token to the `userInfo` endpoint, the authorization server returns a response body that contains the user attributes that are within the boundaries set by the access token scopes. Your application can retrieve information about a user from the `userInfo` endpoint as long as it holds a valid access token with at least an `openid` scope claim.

The `userInfo` endpoint is an OpenID Connect (OIDC) userInfo endpoint. It responds with user attributes when service providers present access tokens that your token endpoint issued. The scopes in your user's access token define the user attributes that the userInfo endpoint returns in its response. The `openid` scope must be one of the access token claims.

Amazon Cognito issues access tokens in response to user pools API requests like InitiateAuth. Because they don't contain any scopes, the userInfo endpoint doesn't accept these access tokens. Instead, you must present access tokens from your token endpoint.

Your OAuth 2.0 third-party identity provider (IdP) also hosts a userInfo endpoint. When your user authenticates with that IdP, Amazon Cognito silently exchanges an authorization code with the IdP `token` endpoint. Your user pool passes the IdP access token to authorize retrieval of user information from the IdP `userInfo` endpoint.

The scopes in a user's access token are determined by the `scopes` request parameter in authentication requests, or the scopes that the pre token generation Lambda trigger adds. You can decode access tokens and examine `scope` claims to see the access-control scopes that they contain. The following are some scope combinations that influence the data returned from the `userInfo` endpoint. The reserved Amazon Cognito scope `aws.cognito.signin.user.admin` has no effect on the data returned from this endpoint.

Example scopes in access token and their effect on the `userInfo` response

`openid`

Returns a response with all user attributes that the app client can read.

`openid profile`

Returns the user attributes `name`, `family_name`, `given_name`, `middle_name`, `nickname`, `preferred_username`, `profile`, `picture`, `website`, `gender`, `birthdate`, `zoneinfo`, `locale`, and `updated_at`. Also returns custom attributes. In app clients that don't have read access to each attribute, the response to this scope is all of the attributes within the specification that your app client does have read access to.

`openid email`

Returns basic profile information and the `email` and `email_verified` attributes.

`openid phone`

Returns basic profile information and the `phone_number` and `phone_number_verified` attributes.

GET /oauth2/userInfo

Your application generates requests to this endpoint directly, not through a browser.

For more information, see UserInfo Endpoint in the OpenID Connect (OIDC) specification.

Topics

- Request parameters in header
- Example – request
- Example – positive response
- Example negative responses

Request parameters in header

`Authorization: Bearer <access_token>`

Pass the access token in the authorization header field.

Required.

Example – request

```
GET /oauth2/userInfo HTTP/1.1
Content-Type: application/x-amz-json-1.1
Authorization: Bearer eyJra12345EXAMPLE
User-Agent: [User agent]
Accept: */*
Host: auth.example.com
Accept-Encoding: gzip, deflate, br
Connection: keep-alive
```

Example – positive response

```
HTTP/1.1 200 OK
Content-Type: application/json;charset=UTF-8
Content-Length: [Integer]
Date: [Timestamp]
x-amz-cognito-request-id: [UUID]
X-Content-Type-Options: nosniff
X-XSS-Protection: 1; mode=block
Cache-Control: no-cache, no-store, max-age=0, must-revalidate
Pragma: no-cache
Expires: 0
Strict-Transport-Security: max-age=31536000 ; includeSubDomains
X-Frame-Options: DENY
Server: Server
Connection: keep-alive
{
    "sub": "[UUID]",
    "email_verified": "true",
    "custom:mycustom1": "CustomValue",
    "phone_number_verified": "true",
    "phone_number": "+12065551212",
    "email": "bob@example.com",
```

```
    "username": "bob"
}
```

For a list of OIDC claims, see Standard Claims. Currently, Amazon Cognito returns the values for `email_verified` and phone_number_verified as strings.

Example negative responses

Example – bad request

```
HTTP/1.1 400 Bad Request
WWW-Authenticate: error="invalid_request",
error_description="Bad OAuth2 request at UserInfo Endpoint"
```

`invalid_request`

The request is missing a required parameter, it includes an unsupported parameter value, or it is otherwise malformed.

Example – bad token

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: error="invalid_token",
error_description="Access token is expired, disabled, or deleted, or the user has
 globally signed out."
```

`invalid_token`

The access token is expired, revoked, malformed, or it's invalid.

The token revocation endpoint

Users who hold a refresh token in their session have something similar to a browser cookie. They can renew their existing session as long as the refresh token is valid. Instead of prompting a user to sign in after their ID or access token expires, your application can use the refresh token to get new, valid tokens. However, you might externally determine that a user's session should be ended, or the user might elect to forget their current session. At that point, you can revoke that refresh token so that they can no longer persist their session.

The `/oauth2/revoke` endpoint revokes a user's access token that Amazon Cognito initially issued with the refresh token that you provide. This endpoint also revokes the refresh token itself and all subsequent access and identity tokens from the same refresh token. After the endpoint revokes the tokens, you can't use the revoked access tokens to access APIs that Amazon Cognito tokens authenticate.

POST /oauth2/revoke

The `/oauth2/revoke` endpoint only supports `HTTPS POST`. The user pool client makes requests to this endpoint directly and not through the system browser.

Request parameters in header

`Authorization`

If your app client has a client secret, the application must pass its `client_id` and `client_secret` in the authorization header through Basic HTTP authorization. The secret is Basic `Base64Encode(client_id:client_secret)`.

`Content-Type`

Must always be `'application/x-www-form-urlencoded'`.

Request parameters in body

`token`

(Required) The refresh token that the client wants to revoke. The request also revokes all access tokens that Amazon Cognito issued with this refresh token.

Required.

`client_id`

(Optional) The app client ID for the token that you want to revoke.

Required if the client is public and doesn't have a secret.

Revocation request examples

This revocation request revokes a refresh token for an app client that has no client secret. Note the `client_id` parameter in the request body.

```
POST /oauth2/revoke HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
token=2YotnFZFEjr1zCsicMWpAA&
client_id=1example23456789
```

This revocation request revokes a refresh token for an app client that *has* a client secret. Note the `Authorization` header that contains an encoded client ID and client secret, but no `client_id` in the request body.

```
POST /oauth2/revoke HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.com
Accept: application/json
Content-Type: application/x-www-form-urlencoded
Authorization: Basic czZCaGRSa3F0MzpnWDFmQmF0M2JW
token=2YotnFZFEjr1zCsicMWpAA
```

Revocation error response

A successful response contains an empty body. The error response is a JSON object with an `error` field and, in some cases, an `error_description` field.

Endpoint errors

- If the token isn't present in the request or if the feature is disabled for the app client, you receive an HTTP 400 and error `invalid_request`.
- If the token that Amazon Cognito sent in the revocation request isn't a refresh token, you receive an HTTP 400 and error `unsupported_token_type`.
- If the client credentials aren't valid, you receive an HTTP 401 and error `invalid_client`.
- If the token has been revoked or if the client submitted a token that isn't valid, you receive an HTTP 200 OK.

The IdP SAML assertion endpoint

The `/saml2/idpresponse` receives SAML assertions. In service-provider-initiated (SP-initiated) sign-in, your application doesn't interact directly with this endpoint—your SAML 2.0 identity

provider (IdP) redirects your user here with their SAML response. For SP-initiated sign-in, configure your IdP with the path to your `saml2/idpresponse` as the assertion consumer service (ACS) URL. For more information about session initiation, see SAML session initiation in Amazon Cognito user pools.

In IdP-initiated sign-in, invoke requests to this endpoint in your application after you sign in user with your SAML 2.0 provider. Your users sign in with your IdP in their browser, then your application collects the SAML assertion and submits it to this endpoint. You must submit SAML assertions in the body of a `HTTP POST` request over HTTPS. The body of your `POST` request must be a `SAMLResponse` parameter and a `Relaystate` parameter. For more information, see Implement IdP-initiated SAML sign-in.

The `saml2/idpresponse` endpoint can accept SAML assertions of up to 100,000 characters in length.

POST /saml2/idpresponse

To use the `/saml2/idpresponse` endpoint in an IdP-initiated sign-in, generate a POST request with parameters that provide your user pool with information about your user's session.

- The app client that they want to sign in to.
- The callback URL that they want to end up at.
- The OAuth 2.0 scopes that they want to request in your user's access token.
- The IdP that initiated the sign-in request.

IdP-initiated request body parameters

SAMLResponse

A Base64-encoded SAML assertion from an IdP associated with a valid app client and IdP configuration in your user pool.

RelayState

A `RelayState` parameter contains the request parameters that you would otherwise pass to the `oauth2/authorize` endpoint. For detailed information about these parameters, see Authorize endpoint.

response_type

The OAuth 2.0 grant type.

client_id

The app client ID.

redirect_uri

The URL where the authentication server redirects the browser after Amazon Cognito authorizes the user.

identity_provider

The name of the identity provider where you want to redirect your user.

idp_identifier

The identifier of the identity provider where you want to redirect your user.

scope

The OAuth 2.0 scopes that you want your user to request from the authorization server.

Example requests with positive responses

Example – POST request

The following request is for an authorization code grant for a user from IdP `MySAMLIdP` in app client `1example23456789`. The user redirects to `https://www.example.com` with their authorization code, which can be exchanged for tokens that include an access token with the OAuth 2.0 scopes `openid`, `email`, and phone.

```
POST /saml2/idpresponse HTTP/1.1
User-Agent: USER_AGENT
Accept: */*
Host: example.auth.us-east-1.amazoncognito.com
Content-Type: application/x-www-form-urlencoded

SAMLResponse=[Base64-encoded SAML assertion]&RelayState=identity_provider
%3DMySAMLIdP%26client_id%3D1example23456789%26redirect_uri%3Dhttps%3A%2F
%2Fwww.example.com%26response_type%3Dcode%26scope%3Demail%2Bopenid%2Bphone
```

Example – response

The following is the response to the previous request.

```
HTTP/1.1 302 Found
Date: Wed, 06 Dec 2023 00:15:29 GMT
Content-Length: 0
x-amz-cognito-request-id: 8aba6eb5-fb54-4bc6-9368-c3878434f0fb
Location: https://www.example.com?code=[Authorization code]
```

OAuth 2.0 grants

The Amazon Cognito user pool OAuth 2.0 authorization server issues tokens in response to three types of OAuth 2.0 authorization grants. You can set the supported grant types for each app client in your user pool. You can't enable *client credentials* grants in the same app client as either *implicit* or *authorization code* grants. Requests for implicit and authorization code grants begin at your Authorize endpoint and requests for client credentials grants start at your Token endpoint.

Authorization code grant

In response to your successful authentication request, the authorization server appends an authorization code in a `code` parameter to your callback URL. You must then exchange the code for ID, access, and refresh tokens with the Token endpoint. To request an authorization code grant, set `response_type` to `code` in your request. For an example request, see Example: authorization code grant. Amazon Cognito supports Proof Key for Code Exchange (PKCE) in authorization code grants.

The authorization code grant is the most secure form of authorization grant. It doesn't show token contents directly to your users. Instead, your app is responsible for retrieving and securely storing your user's tokens. In Amazon Cognito, an authorization code grant is the only way to get all three token types—ID, access, and refresh—from the authorization server. You can also get all three token types from authentication through the Amazon Cognito user pools API, but the API doesn't issue access tokens with scopes other than `aws.cognito.signin.user.admin`.

Implicit grant

In response to your successful authentication request, the authorization server appends an access token in an `access_token` parameter, and an ID token in an `id_token` parameter, to your callback URL. An implicit grant requires no additional interaction with the Token endpoint. To request an implicit grant, set `response_type` to `token` in your request. The implicit grant only generates an ID and access token. For an example request, see Example: Token (implicit) grant without `openid` scope.

The implicit grant is a legacy authorization grant. Unlike with the authorization code grant, users can intercept and inspect your tokens. To prevent token delivery through implicit grant, configure your app client to support authorization code grant only.

Client credentials

Client credentials is an authorization-only grant for machine-to-machine access. To receive a client credentials grant, bypass the Authorize endpoint and generate a request directly to the Token endpoint. Your app client must have a client secret and support client credentials grants only. In response to your successful request, the authorization server returns an access token.

The access token from a client credentials grant is an authorization mechanism that contains OAuth 2.0 scopes. Typically, the token contains custom scope claims that authorize HTTP operations to access-protected APIs. For more information, see Scopes, M2M, and resource servers.

Client credentials grants add costs to your AWS bill. For more information, see Amazon Cognito Pricing.

Refresh token

You can request a refresh token grant directly from the Token endpoint. This grant returns new ID and access tokens in exchange for a valid refresh token.

For more perspective on these grants and their implementation, see How to use OAuth 2.0 in Amazon Cognito: Learn about the different OAuth 2.0 grants in the *AWS Security Blog*.

Using PKCE in authorization code grants

Amazon Cognito supports Proof Key for Code Exchange (PKCE) authentication in authorization code grants. PKCE is an extension to the OAuth 2.0 authorization code grant for public clients. PKCE guards against the redemption of intercepted authorization codes.

How Amazon Cognito uses PKCE

To start authentication with PKCE, your application must generate a unique string value. This string is the code verifier, a secret value that Amazon Cognito uses to compare the client requesting the initial authorization grant to the client exchanging the authorization code for tokens.

Your app must apply an SHA256 hash to the code verifier string and encode the result to base64. Pass the hashed string to the Authorize endpoint as a `code_challenge` parameter in the request

body. When your app exchanges the authorization code for tokens, it must include the code verifier string in plaintext as a `code_verifier` parameter in the request body to the Token endpoint. Amazon Cognito performs the same hash-and-encode operation on the code verifier. Amazon Cognito only returns ID, access, and refresh tokens if it determines that the code verifier results in the same code challenge that it received in the authorization request.

To implement Authorization Grant Flow with PKCE

1. Open the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool. If you create a user pool, you will be prompted to set up an app client and configure managed login during the wizard.

   a. If you create a new user pool, set up an app client and configure managed login during the guided setup.

   b. If you configure an existing user pool, add a domain and a public app client, if you haven't already.
4. Generate a random alphanumeric string, typically a universally unique identifier (UUID), to create a code challenge for the PKCE. This string is the value of the `code_verifier` parameter that you will submit in your request to the Token endpoint.
5. Hash the `code_verifier` string with the SHA256 algorithm. Encode the result of the hashing operation to base64. This string is the value of the `code_challenge` parameter that you will submit in your request to the Authorize endpoint.

   The following Python example generates a `code_verifier` and calculates the `code_challenge`:

```
#!/usr/bin/env python3

import secrets
from base64 import urlsafe_b64encode
from hashlib import sha256
from string import ascii_letters
from string import digits

# use the secrets module for cryptographically strong random values
alphabet = ascii_letters + digits
code_verifier = ''.join(secrets.choice(alphabet) for _ in range(128))
code_verifier_hash = sha256(code_verifier.encode()).digest()
```

```
code_challenge = urlsafe_b64encode(code_verifier_hash).decode().rstrip('=')

print(f"code challenge: {code_challenge}")
print(f"code verifier: {code_verifier}")
```

The following is an example output from the Python script:

```
code challenge: Eh0mg-OZv7BAyo-tdv_vYamx1boOYDulDklyXoMDtLg
code verifier: 9D-aW_iygXrgQcWJd0y0tNVMPSXSChIc2xceDhvYVdGLCBk-
JWFTmBNjvKSdOrjTTYazOFbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFGyE8yA-05-_v7Dxf3EIYJH
```

6. Complete managed login sign-in with an authorization code grant request with PKCE. The following is an example URL:

```
https://mydomain.auth.us-east-1.amazoncognito.com/oauth2/authorize?
response_type=code&client_id=1example23456789&redirect_uri=https://
www.example.com&code_challenge=Eh0mg-OZv7BAyo-
tdv_vYamx1boOYDulDklyXoMDtLg&code_challenge_method=S256
```

7. Collect the authorization code and redeem it for tokens with the token endpoint. The following is an example request:

```
POST /oauth2/token HTTP/1.1
Host: mydomain.auth.us-east-1.amazoncognito.com
Content-Type: application/x-www-form-urlencoded
Content-Length: 296

redirect_uri=https%3A%2F%2Fwww.example.com&
client_id=1example23456789&
code=7378f445-c87f-400c-855e-0297d072ff03&
grant_type=authorization_code&
code_verifier=9D-aW_iygXrgQcWJd0y0tNVMPSXSChIc2xceDhvYVdGLCBk-
JWFTmBNjvKSdOrjTTYazOFbUmrFERrjWx6oKtK2b6z_x4_gHBDlr4K1mRFGyE8yA-05-_v7Dxf3EIYJH
```

8. Review the response. It will contain ID, access, and refresh tokens. For more information about using Amazon Cognito user pool tokens, see Understanding user pool JSON web tokens (JWTs).

Managed login and federation error responses

A sign-in process in managed login or federated sign-in might return an error. The following are some conditions that can cause authentication to end with an error.

- A user performs an operation that your user pool can't fulfill.
- A Lambda trigger doesn't respond with expected syntax.
- Your identity provider (IdP) returns an error.
- Amazon Cognito couldn't validate attribute information that your user provided.
- Your IdP didn't send claims that map to required attributes.

When Amazon Cognito encounters an error, it communicates it in one of the following ways.

1. Amazon Cognito sends a redirect URL with the error in the request parameters.
2. Amazon Cognito displays an error in managed login.

Errors that Amazon Cognito appends to request parameters have the following format.

```
https://<Callback URL>/?error_description=error+description&error=error+name
```

When you help your users submit error information when they can't perform an operation, request that they capture the URL *and* the text or a screenshot of the page.

> **Note**
>
> Amazon Cognito error descriptions are not fixed strings and you shouldn't use logic that relies on a fixed pattern or format.

OIDC and social identity provider error messages

Your identity provider might return an error. When an OIDC or OAuth 2.0 IdP returns an error that conforms to standards, Amazon Cognito redirects your user to the callback URL and adds the provider error response to error request parameters. Amazon Cognito adds the provider name and HTTP error code to the existing error strings.

The following URL is an example redirect from an IdP that returned an error to Amazon Cognito.

```
https://www.amazon.com/?error_description=LoginWithAmazon+Error+-+400+invalid_request
+The+request+is+missing+a+required+parameter+%3A+client_secret&error=invalid_request
```

Because Amazon Cognito only returns what it receives from a provider, your user might see a subset of this information.

When your user encounters an issue with initial sign-in through your IdP, the IdP delivers any error messages directly to your user. Amazon Cognito relays an error message to your user when it generates a request to your IdP to validate your user's session. Amazon Cognito relays OAuth and OIDC IdP error messages from the following endpoints.

`/token`

Amazon Cognito exchanges an IdP authorization code for an access token.

`/.well-known/openid-configuration`

Amazon Cognito discovers the path to your issuer endpoints.

`/.well-known/jwks.json`

To verify your user's JSON Web Tokens (JWTs), Amazon Cognito discovers the JSON Web Keys (JWKs) that your IdP uses to sign tokens.

Because Amazon Cognito doesn't initiate outbound sessions to SAML 2.0 providers that might return HTTP errors, your users' errors during a session with a SAML 2.0 IdP don't include this form of provider error message.

Deploying and operating

Once your Amazon Cognito design is in place, you operate it as part of a production service. This section covers the operational concerns of running Amazon Cognito at scale: request-rate and resource quotas, provisioned capacity, multi-Region replication, resource tagging, and troubleshooting. For logging and monitoring—CloudWatch metrics and CloudTrail event history—see Logging and monitoring in Amazon Cognito in the Security chapter.

Topics

- Quotas in Amazon Cognito
- Multi-Region replication for user pools
- Tagging Amazon Cognito resources
- Troubleshoot Amazon Cognito

Quotas in Amazon Cognito

Amazon Cognito has default quotas, formerly referred to as *limits*, for the maximum number of operations that you can perform in your account. Amazon Cognito also has quotas for the maximum number and size of Amazon Cognito resources.

Each Amazon Cognito quota represents a maximum volume of requests in one AWS Region in one AWS account. For example, your apps can make API requests at *up to* the **Default quota (RPS)** rate for `UserAuthentication` operations against all of your user pools in US East (N. Virginia). Your apps in Asia Pacific (Tokyo) can produce the same volume of requests against all of your user pools in their own Region. AWS can only grant a quota increase request in one Region at a time. A successful quota increase in US East (N. Virginia) has no effect on your maximum request rate in Asia Pacific (Tokyo).

Topics

- Understanding API request rate quotas
- Managing API request rate quotas
- Amazon Cognito user pools API operation categories and request rate quotas
- Amazon Cognito identity pools (federated identities) API operation request rate quotas
- Quotas on resource number and size

Understanding API request rate quotas

Quota categorization

Amazon Cognito enforces a maximum request rate for API operations. For more information about the API operations that Amazon Cognito makes available, see the API reference guides for user pools and identity pools. For user pools, these operations are grouped into categories of common use cases like `UserAuthentication` or `UserCreation`. For a list of user pool API operations by category, see Amazon Cognito user pools API operation categories and request rate quotas.

In the Service Quotas console, you can track your quota usage by category user pools and identity pools.If the request rate of your Amazon Cognito user pools or exceeds a quota, you can purchase additional capacity. You can track your user pool quota usage by category and purchase quota increases in the Service Quotas console.

Operation quotas are defined as the maximum number of requests per second (RPS) for all operations within a category. The Amazon Cognito user pools service applies quotas to all operations in each category. For example, the category `UserCreation` includes four operations: `SignUp`, `ConfirmSignUp`, `AdminCreateUser`, and `AdminConfirmSignUp`. It's allocated with a combined quota of 50 RPS. If multiple operations take place at the same time, each operation within this category can call up to 50 RPS separately or combined.

> **Note**
>
> Category quotas only apply to user pools. Amazon Cognito applies each identity pool quota to a single operation. For both per-category and per-operation request rate quotas, AWS measures the aggregate rate of all requests from all user pools or identity pools in your AWS account in one Region.

Amazon Cognito user pools API operations with special request rate handling

Operation quotas are measured and enforced for the combined total requests at the category level, except for the `AdminRespondToAuthChallenge` and `RespondToAuthChallenge` operations, where special handling rules are applied.

The `UserAuthentication` category includes four operations in the Amazon Cognito user pools API: `AdminInitiateAuth`, `InitiateAuth`, `AdminRespondToAuthChallenge`, and

RespondToAuthChallenge. Additionally, user authentication in the hosted UI contributes to this quota. The InitiateAuth and AdminInitiateAuth operations are measured and enforced per category quota. The matching operations RespondToAuthChallenge and AdminRespondToAuthChallenge are subject to a separate quota that is three times the UserAuthentication category limit. This elevated quota accommodates multiple authentication challenges set up in your apps. The quota is sufficient to cover the large majority of use cases. After your app makes up to three responses to authentication challenges, additional requests count toward the UserAuthentication category quota. Multi-factor authentication (MFA), device authentication, and custom authentication are all examples of challenge prompts that you might engineer into your user pool.

For example, if your quota for the UserAuthentication category is 80 RPS, you can call RespondToAuthChallenge or AdminRespondToAuthChallenge at a rate up to 240 RPS (3 * 80 RPS). If your user pool prompts for four rounds of challenge per authentication and 70 users sign in per second, then the total RespondToAuthChallenge is 280 RPS (70 x 4), which is 40 RPS above the quota. The extra 40 RPS is added to 70 InitiateAuth calls, making the total usage of UserAuthentication category 110 RPS (40 + 70). Because this value exceeds the category quota set at 80 RPS by 30 RPS, Amazon Cognito throttles requests from your app.

Monthly active users

When Amazon Cognito calculates user pool billing, it charges you a rate for each *monthly active user (MAU)*. Consider your current and projected MAU count in your planning for quota increase requests. A user is counted as a MAU if, within a calendar month, there is an identity operation related to that user. When you link federated users to local users, with SAML or OIDC federation, the local user will count as an enterprise directory MAU or EnterpriseMAU, regardless of whether the user signs in directly or via federation. See Amazon Cognito Pricing for more information.

- Sign-up or administrative creation of a user. User CSV import *doesn't* contribute to your MAU count.
- User account confirmation or attribute verification.
- Sign-in and challenge response. Operations that you authorize with the currently signed-in user's access token don't contribute to your MAU count; however, because sign-in produces access tokens, these operations indicate that the associated user is an MAU.
- Sign-out and token revocation.
- Password self-service reset and setting of user passwords as an administrator. *Resetting* user passwords as an administrator (AdminResetUserPassword) doesn't contribute to your MAU count.

- Change user attributes or group membership.
- Query detailed attributes of a user as an administrator.

> **Note**
>
> The category *Query detailed attributes of a user as an administrator* includes the API operation AdminGetUser, but not ListUsers. A detailed user-by-user query in a large user pool can have a significant impact on your AWS bill. To avoid additional cost, collect user data with `ListUsers` or store user information in an external database.

You aren't charged for additional sessions by any active user, or for any users that weren't active within a calendar month. In a month where you have changed your user pool feature plan between the available options of *Lite*, *Essentials*, and *Plus*, your bill for that month is computed from the sum of monthly active users (MAUs) in each tier, with each MAU assigned to the highest-priced assigned tier when the user was active. For example:

1. At the beginning of the month, your user pool is on the Plus feature plan.
2. User A signs in on the first day of the month.
3. User B signs in on the first and last days of the month.
4. On the tenth day of the month, you switch your feature plan to Essentials.
5. User C signs in on the last day of the month.

In this scenario, user A and user B are Plus MAUs and user C is an Essentials MAU.

Lite MAU

A user that was active at least once in a month when the user pool was on the Lite feature plan, and was never active when the user pool was on the Essentials or Plus plans.

Essentials MAU

A user that was active at least once in a month when the user pool was on the Essentials feature plan, and was never active when the user pool was on the Plus plan.

Plus MAU

A user that was active at least once in a month when the user pool was on the Plus plan.

For more information, see User pool feature plans.

Managing API request rate quotas

Identify quota requirements

> **Important**
>
> If you increase Amazon Cognito quotas for categories such as `UserAuthentication`, `UserCreation`, or `AccountRecovery`, you may need to increase quotas for other AWS services. For example, messages that Amazon Cognito sends with Amazon Simple Notification Service (Amazon SNS) or Amazon Simple Email Service (Amazon SES) can fail if request rate quotas are insufficient in those services.

To calculate quota requirements, determine how many active users will interact with your application in a specific time period. For example, if you expect your application to sign in an average of one million active users within an eight-hour period, then you must be able to authenticate an average of 35 users per second.

In addition, if you assume that the average user session is two hours, and you configure tokens to expire after an hour, each user must refresh their tokens once during their session. The required average quota for the `UserAuthentication` category to support this load is 70 RPS.

If you assume a peak-to-average ratio of 3:1 by accounting for the variance of user sign-in frequency during the eight-hour period, then you need the desired `UserAuthentication` quota of 200 RPS.

> **Note**
>
> If you call multiple operations for each user action, you must sum up the individual operation call rates at the category level.

Optimize request rates for quota limits

Because increasing API rate limits adds costs to your AWS bill, consider adjustments to your usage model before you request a quota increase. The following are some examples of app architecture that optimizes request rates.

Retry the attempt after a back-off waiting period

You can catch errors with each API call, and then re-try the attempt after a back-off period. You can adjust the back-off algorithm according to business needs and load. Amazon SDKs have built-in retry logic. For more information, see Tools to Build on AWS.

Use an external database for frequently updated attributes

If your application requires several calls to a user pool to read or write custom attributes, use external storage. You can use your preferred database to store custom attributes or use a cache layer to load a user profile during sign-in. You can reference this profile from the cache when needed, instead of reloading the user profile from a user pool.

Validate JSON web tokens (JWTs) on the client side

Applications must validate JWT tokens before trusting them. You can verify the signature and validity of tokens on the client side without sending API requests to a user pool. After the token is validated, you can trust claims in the token and use the claims instead of making more `getUser` API calls. For more information, see Verifying a JSON Web Token.

Throttle traffic to your web application with a waiting room

If you expect traffic from a large number of users signing in during a time-bound event, such as taking an exam or attending a live event, you can optimize request traffic with self-throttling mechanisms. You can, for example, set up a waiting room where users can stand by until a session is available, allowing you to process requests when you have available capacity. See the AWS Virtual Waiting Room solution for a reference architecture of a waiting room.

Cache JWTs

Reuse access tokens until they expire. For an example framework with token caching in an API Gateway, see Managing user pool token expiration and caching. Instead of generating API requests to query user information, cache ID tokens until they expire, and read user attributes from the cache.

For more information about working with API request rates in AWS, see Managing and monitoring API throttling in your workloads. For information about optimizing Amazon Cognito operations that add costs to your AWS bill, see Managing costs.

Track quota usage

Amazon Cognito generates `CallCount` and `ThrottleCount` metrics in Amazon CloudWatch for each API operation category at the account level. You can use `CallCount` to track the total number of calls customers made related to a category. You can use `ThrottleCount` to track the total number of throttled calls related to a category. You can use the `CallCount` and `ThrottleCount` metrics with the `Sum` statistic to count the total number of calls in a category. For more information, see CloudWatch usage metrics.

When monitoring service quotas, *utilization* is the percentage of a service quota in use. For example, if the quota value is 200 resources, and 150 resources are in use, the utilization is 75%. *Usage* is the number of resources or operations in use for a service quota.

Tracking usage through CloudWatch metrics

You can track and collect Amazon Cognito user pools utilization metrics with CloudWatch. The CloudWatch dashboard displays metrics about every AWS service that you use. With CloudWatch, you can create metric alarms to notify you or change a specific resource that you are monitoring. For more information about CloudWatch metrics, see Track your CloudWatch usage metrics.

Tracking utilization through Service Quotas metrics

Amazon Cognito user pools are integrated with Service Quotas, a console interface to display and manage your service quota usage. In the Service Quotas console, you can look up the value of a specific quota, view monitoring information, request a quota increase, or set up CloudWatch alarms. After your account has been active for a while, you can view a graph of your resource utilization.

The **Applied account-level quota value** column in the Service Quotas console for Amazon Cognito user pools and Amazon Cognito identity pools displays your current quota. The **Utilization** column displays your current rate of quota usage. Adjustable Amazon Cognito user pools requests-per-second (RPS) quotas display their current usage. The Service Quotas console can also navigate you to CloudWatch metrics for a closer look at a selected quota metric. For more information on viewing quotas in the Service Quotas console, see Viewing Service Quotas.

Track monthly active users (MAUs)

The number of monthly active users (MAUs) in your user pool contributes important data to your planning for increases to request-rate quotas. You can compare your API request rates to the number of users you had active in a given time period. With that knowledge, you can calculate how

an increase in active users of your applications will affect your quotas in your usage model. For example, imagine that your combined applications in US West (Oregon) resulted in 2 million active users in a month and your `UserAuthentication` category received occasional throttling errors at the default quota of 120 requests per second (RPS). In the previous month, before your successful advertising campaign, you had 1 million MAUs and your applications never exceeded 80 RPS. If you anticipate a similar spike as a result of a new TV spot, you might purchase an additional 40 RPS to accommodate the next million users with an adjusted quota of 160 RPS.

To review your MAUs

Access the AWS Billing console and review a recent bill. Under **charges by service**, you can filter on **Cognito** to view a breakdown of your MAUs for that billing period.

Requesting a quota increase

Amazon Cognito has a quota for the maximum number of operations per second that you can perform in your user pools and identity pools in each AWS Region. You can purchase an increase to adjustable Amazon Cognito user pools API request rate quotas. For adjustable user pools categories, you can request a higher account-level max limit in Service Quotas and then use the Amazon Cognito provisioning API to set your desired rate. For more information, see Managing provisioned limits.

To request a higher account-level max limit, use the Service Quotas console or the Service Quotas API operations `ListAWSDefaultServiceQuotas` and `RequestServiceQuotaIncrease`.

- To request an account-level max limit increase using the Service Quotas console, see Requesting a API quota increase in the *Service Quotas User Guide*.
- Some account-level max limit increases are auto-approved. Otherwise, AWS targets completion of quota increase requests within 10 days. However, several considerations might cause the request processing time to exceed 10 days. Some requests, for example, might require Amazon Cognito to provision additional hardware capacity, and seasonal increases in request volumes might introduce delays.
- If the quota isn't available in Service Quotas, use the Service limit increase form.

> **Note**
>
> Only adjustable quotas can be increased.

Note

Quota increases are regional configurations. A quota increase in one AWS Region doesn't affect your quotas in other Regions. If you require higher API quotas in multiple Regions, you must request a quota increase in each Region separately.

Note

Requesting an account-level max limit increase is only the first step. After your request is approved, you must call `UpdateProvisionedLimit` or use the Amazon Cognito console to set your provisioned limit. For more information, see Managing provisioned limits.

Managing provisioned limits

Amazon Cognito user pools supports a two-tier quota model for adjustable API rate quotas. With this model, you can programmatically increase or decrease your provisioned request rate without contacting AWS Support. The two tiers work as follows:

Account-level max limit

The maximum rate that you can provision, set through Service Quotas. To increase your account-level max limit, submit a quota increase request in the Service Quotas console. Some requests are auto-approved. For more information, see Requesting a quota increase.

Provisioned limit

The rate, in requests per second (RPS), that Amazon Cognito currently enforces for your account. You can adjust this value up to your account-level max limit with the Amazon Cognito provisioning API operations.

For example, if your `UserAuthentication` account-level max limit is 500 RPS, you can use the `UpdateProvisionedLimit` API operation to set your provisioned limit to 300 RPS. You can later reduce the provisioned limit back to the default, and you are only billed for the time at the higher rate.

> **Important**
>
> You are billed for provisioned capacity above the default rate, prorated based on actual usage time. For pricing, see Amazon Cognito pricing.

Managing provisioned limits in the console

You can view and manage your provisioned limits in the Amazon Cognito console. Open the **User pools** page and choose the **Provisioned limits** tab. On this tab, you can view your current provisioned limits and account-level max limits for each adjustable API category. You can increase or decrease your provisioned rate within your approved account-level max limit.

Provisioning API operations

Amazon Cognito provides the following API operations for managing provisioned limits. These operations require authentication and the corresponding IAM permissions.

GetProvisionedLimit

Returns the current provisioned limit value for a specific API category. The response includes the provisioned limit value and the default (free) limit value.

UpdateProvisionedLimit

Sets your provisioned limit value for a specific API category. The requested value must be between the default limit and your Service Quotas account-level max limit. This operation takes effect immediately.

Both operations take a `LimitDefinition` parameter that identifies the limit to manage. For user pools API rate limits, the limit definition has a class of `API_CATEGORY` and an attributes map with the `Category` key. For example, to manage the `UserAuthentication` limit, specify a limit definition with `{"Category": "UserAuthentication"}`.

Example GetProvisionedLimit request and response

The following example returns the provisioned limit for the `UserAuthentication` API category.

Request

```
{
```

```
    "LimitDefinition": {
        "LimitClass": "API_CATEGORY",
        "Attributes": {"Category": "UserAuthentication"}
    }
}
```

Response

```
{
    "Limit": {
        "LimitDefinition": {
            "LimitClass": "API_CATEGORY",
            "Attributes": {"Category": "UserAuthentication"}
        },
        "ProvisionedLimitValue": 120,
        "FreeLimitValue": 120
    }
}
```

Example UpdateProvisionedLimit request and response

The following example sets the provisioned limit for the `UserAuthentication` API category to 300 RPS.

Request

```
{
    "LimitDefinition": {
        "LimitClass": "API_CATEGORY",
        "Attributes": {"Category": "UserAuthentication"}
    },
    "RequestedLimitValue": 300
}
```

Response

```
{
    "Limit": {
        "LimitDefinition": {
            "LimitClass": "API_CATEGORY",
            "Attributes": {"Category": "UserAuthentication"}
```

```
        },
        "ProvisionedLimitValue": 300,
        "FreeLimitValue": 120
    }
}
```

The following IAM actions control access to these operations:

- `cognito-idp:GetProvisionedLimit`
- `cognito-idp:UpdateProvisionedLimit`

> **Note**
>
> Cross-account access is not supported. Only the owning account can manage its provisioned limits.

Provisioning workflow

The following workflow describes how to increase your provisioned limit.

1. Determine your required rate. For more information, see Identify quota requirements.
2. If your required rate exceeds your current Service Quotas account-level max limit, request an increase in the Service Quotas console.
3. After your account-level max limit is approved, call `UpdateProvisionedLimit` with your desired rate.
4. To reduce your provisioned rate and stop additional billing, call `UpdateProvisionedLimit` with the default limit value or a lower value.

Considerations

- Provisioned limits are account-level resources. They apply to the aggregate rate of all requests from all user pools in one AWS Region in your AWS account.
- The `LimitManagement` category, which includes `GetProvisionedLimit` and `UpdateProvisionedLimit`, is rate-limited to 1 request per second per account.
- Only adjustable quota categories support provisioning. For a list of adjustable categories, see Amazon Cognito user pools API operation categories and request rate quotas.

- If your user pool uses managed login, you can't use the provisioning API to adjust the `UserAuthentication` or `UserFederation` categories. To increase these quotas, submit a Service Quotas increase request through the Service Quotas console or use the Service limit increase form.
- Changes to provisioned quotas are regional. A provisioned quota change in one AWS Region doesn't affect your quotas in other Regions.
- All calls to provisioning API operations are logged in AWS CloudTrail.

Amazon Cognito user pools API operation categories and request rate quotas

Because Amazon Cognito has overlapping classes of API operations with differing authorization models, each operation belongs to a category. Each category has its own pooled quota for all member API operations, across all user pools in one AWS Region in your account. You can only request an increase to *adjustable* category quotas. For more information, see Requesting a quota increase. Quota adjustments apply to the user pools in your account in a single Region. Amazon Cognito restricts operations in some categories[3] to 5 requests per second (RPS), per user pool. The **Default quota (RPS)** additionally applies to all user pools in an AWS account.

> **Note**
>
> The quota for each category is measured in Monthly Active Users (MAUs). AWS accounts with fewer than two million MAUs can operate within the default quota. If you have less than one million MAUs and Amazon Cognito is throttling requests, consider optimizing your app. For more information, see Optimize request rates for quota limits.

Category operation quotas are applied across all users in all user pools within one AWS Region. Amazon Cognito also maintains a quota for the number of requests that your app can generate against one user. You must limit per-user API requests as shown in the following table.

Amazon Cognito user pools per-user request rate quotas

Operation	Operations per user per second
Read user profile	10

Operation	Operations per user per second
Examples: `GetUser`, `GetDevice` , `InitiateAuth` , `RespondToAuthChallenge`	
Write user profile Examples: `UpdateUserAttributes` , `SetUserSettings`	10

You must limit per-category API requests as shown in the following table.

Amazon Cognito user pools per-category request rate quotas

Category	Description	Default quota (RPS)	Adjustable
`UserAuthentication` • InitiateAuth • Token refresh with `InitiateAuth` or the token endpoint • RespondToAuthChallenge[1] • AdminInitiateAuth • AdminRespondToAuthChallenge[1] • GetTokensFromRefreshToken • Managed login or classic hosted UI sign-in and MFA for local users in	Operations that authenticate (sign in) a user. These operations are subject to Amazon Cognito user pools API operations with special request rate handling.	120	Yes

Category	Description	Default quota (RPS)	Adjustable
authorization-code or implicit grants[2]			
`UserCreation` • SignUp • ConfirmSignUp • AdminCreateUser • AdminConfirmSignUp	Operations that create or confirm an Amazon Cognito *local user*. This is a user that is created and verified directly by your Amazon Cognito user pools.	50	Yes
`UserFederation` Operations that federate (authenticate) users with a third-party identity provider into your Amazon Cognito user pools.	Operations that submit an IdP response to a user pool federation endpoint. OIDC or social provider operations that result in an IdP token, and all SAML requests, contribute to this quota.	25	Yes

Category	Description	Default quota (RPS)	Adjustable
`UserAccountRecovery` • ChangePassword • ConfirmForgotPassword • ForgotPassword • AdminResetUserPassword • AdminSetUserPassword • RespondToAuthChallenge[1] • AdminRespondToAuthChallenge[1] • Managed login password reset	Operations that recover a user's account, or change or update a user's password.	30	No
`UserRead` • AdminGetUser • AdminGetUserAuthFactors • GetUser	Operations that retrieve a user from your user pools.	120	Yes

Category	Description	Default quota (RPS)	Adjustable
`UserUpdate` • AdminAddUserToGroup • AdminDeleteUserAttributes • AdminUpdateUserAttributes • AdminDeleteUser • AdminDisableUser • AdminEnableUser • AdminLinkProviderForUser • AdminDisableProviderForUser • VerifyUserAttribute • DeleteUser • DeleteUserAttributes • UpdateUserAttributes • AdminUserGlobalSignOut • GlobalSignOut • AdminRemoveUserFromGroup	Operations that you use to manage users and user attributes.	25	No
`UserToken` • RevokeToken	Operations for token management	120	Yes

Category	Description	Default quota (RPS)	Adjustable
`UserResourceRead` • AdminGetDevice • AdminListGroupsForUser • AdminListDevices • GetDevice • ListDevices • GetUserAttributeVerificationCode • ResendConfirmationCode • AdminListUserAuthEvents	Operations that retrieve user resource information from Amazon Cognito, such as a remembered device or a group membership.	50	Yes

Category	Description	Default quota (RPS)	Adjustable
`UserResourceUpdate` • AdminForgetDevice • AdminUpdateAuthEventFeedback • AdminSetUserMFAPreference • AdminDeleteSoftwareToken • AdminSetUserSettings • AdminUpdateDeviceStatus • UpdateDeviceStatus • UpdateAuthEventFeedback • ConfirmDevice • SetUserMFAPreference • SetUserSettings • VerifySoftwareToken • AssociateSoftwareToken • ForgetDevice	Operations that update resource information for a user, such as a remembered device or a group membership.	25	No

Category	Description	Default quota (RPS)	Adjustable
`UserList` • ListUsers • ListUsersInGroup	Operations that return a list of users.	30	No
`UserPoolRead` • DescribeUserPool • ListUserPools	Operations that read your user pools.	15	No
`UserPoolUpdate` • CreateUserPool • UpdateUserPool • DeleteUserPool	Operations that create, update, or delete your user pools.	15	No

Category	Description	Default quota (RPS)	Adjustable
`UserPoolResourceRead` • DescribeIdentityProvider • DescribeResourceServer • DescribeUserImportJob • DescribeUserPoolDomain • GetCSVHeader • GetGroup • GetSigningCertificate • GetIdentityProviderByIdentifier • GetUserPoolMfaConfig • ListGroups • ListIdentityProviders • ListResourceServers • ListTagsForResource • ListUserImportJobs • DescribeRiskConfiguration • GetUICustomization	Operations that retrieve information about resources, such as groups or resource servers, from a user pool.[3]	20	No

Category	Description	Default quota (RPS)	Adjustable
• DescribeManagedLoginBranding • DescribeManagedLoginBrandingByClient • ListUserPoolReplicas			

Category	Description	Default quota (RPS)	Adjustable
`UserPoolResourceUpdate` • AddCustomAttributes • CreateGroup • CreateIdentityProvider • CreateResourceServer • CreateUserImportJob • CreateUserPoolDomain • DeleteGroup • DeleteIdentityProvider • DeleteResourceServer • DeleteUserPoolDomain • SetUserPoolMfaConfig • StartUserImportJob • StopUserImportJob • UpdateGroup • UpdateIdentityProvider • UpdateResourceServer	Operations that modify resources, such as groups or resource servers, in a user pool.[3]	15	No

Category	Description	Default quota (RPS)	Adjustable
• UpdateUserPoolDomain • SetRiskConfiguration • SetUICustomization • TagResource • UntagResource • CreateManagedLoginBranding • UpdateManagedLoginBranding • DeleteManagedLoginBranding • CreateUserPoolReplica • UpdateUserPoolReplica • DeleteUserPoolReplica			
`UserPoolClientRead` • DescribeUserPoolClient • ListUserPoolClients	Operations that retrieve information about your user pool clients.[3]	15	No

Category	Description	Default quota (RPS)	Adjustable
`UserPoolClientUpdate` • CreateUserPoolClient • DeleteUserPoolClient • UpdateUserPoolClient	Operations that create, update, and delete your user pool clients.[3]	15	No
`ClientAuthentication` • `client_credentials` grant type requests to the token endpoint • GetClientToken	Operations that generate credentials to be used in authorizing machine-to-machine requests	150	No
`LimitManagement` • GetProvisionedLimit • UpdateProvisionedLimit	Operations that retrieve or update your provisioned API rate limits	1	No

[1] A `RespondToAuthChallenge` or `AdminRespondToAuthChallenge` response with a `ChallengeName` of `NEW_PASSWORD_REQUIRED` counts toward the `UserAccountRecovery` category. All other challenge responses count toward the `UserAuthentication` category.

[2] Each managed login or classic hosted UI operation during sign-in contributes one request to the quota. For example, a user who signs in and provides an MFA code contributes 2 requests. Token redemption in authorization-code grants is subject to an additional quota allocation at the same rate as your quota in the `UserAuthentication` category.

[3] Any individual operation in this category has a constraint that prevents the operation from being called at a rate higher than 5 RPS for a single user pool.

Bulk request-rate limits for user pool domains

The following quotas apply to the overall volume of requests to a user pool domain.

Operation	Description	Default quota (RPS)	Adjustable
Requests from source IP	Volume of requests from one IP address to one domain	300	No
Requests to app client	Volume of requests for one app client ID in one domain	300	No
Requests to domain	Overall volume of requests for the services of one user pool domain	500	No
Requests for JSON web key documents	Volume of requests for `jwks.json` in one AWS account in one AWS Region	50,000	No

Amazon Cognito identity pools (federated identities) API operation request rate quotas

Operation	Description	Default quota (RPS)[1]	Adjustable	Quota increase eligibility
`GetId`	Retrieve an identity ID from an identity pool.	25	Yes	Contact your account team.

Operation	Description	Default quota (RPS)[1]	Adjustable	Quota increase eligibility
`GetOpenIdToken`	Retrieve an OpenID token from an identity pool in the classic workflow.	200	Yes	Contact your account team.
`GetCredentialsForIdentity`	Retrieve AWS credentials from an identity pool in the enhanced workflow.	200	Yes	Contact your account team.
`GetOpenIdTokenForDeveloperIdentity`	Retrieve an OpenID token from an identity pool in the developer workflow.	50	Yes	Contact your account team.
`ListIdentities`	Retrieve a list of identity IDs in an identity pool.	5	Yes	Contact your account team.
`DeleteIdentities`	Delete one or more registered identities from an identity pool.	10	Yes	Contact your account team.
`TagResource`	Apply a tag to an identity pool.	5	Yes	Contact your account team.
`UntagResource`	Remove a tag from an identity pool.	5	Yes	Contact your account team.

Operation	Description	Default quota (RPS)[1]	Adjustable	Quota increase eligibility
`ListTagsForResource`	Display a list of the tags applied to an identity pool.	10	Yes	Contact your account team.

[1] The default quota is the minimum request rate quota for the identity pools in any AWS Region in your AWS account. Your RPS quota might be higher in some Regions.

Quotas on resource number and size

Resource quotas are the maximum number or size of resources, input fields, time duration, and other miscellaneous features in Amazon Cognito.

You can request an adjustment to some resource quotas in the Service Quotas console or from a Service limit increase form. To request a quota from the Service Quotas console, see Requesting a quota increase in the *Service Quotas User Guide*. If the quota isn't available in Service Quotas, use the Service limit increase form.

> **Note**
>
> Resource quotas at the AWS account level, like *User pools per Region*, apply to Amazon Cognito resources in each AWS Region. For example, you can have 1,000 user pools in US East (N. Virginia) and another 1,000 in Europe (Stockholm).

The following tables indicate default resource quotas, and whether they're adjustable.

Amazon Cognito user pools resource quotas

The following quotas describe the maximum number or length of items that you can create in user pools.

Resource	Quota	Adjustable	Maximum quota
App clients per user pool	1,000	Yes	10,000
User pools per Region	1,000	Yes	10,000
Identity providers per user pool	300	Yes	1,000
Resource servers per user pool	100	Yes	300
Users per user pool	40,000,000	Yes	Contact your account team.
Total combined changes in pre token generation Lambda trigger[1]	5,000	Yes	Contact your account team.
Managed login branding styles per user pool	20	No	N/A
Managed login terms documents per user pool	40	No	N/A
Custom attributes per user pool	50	No	N/A
Characters per attribute	2,048 bytes	No	N/A
Characters in custom attribute name	20	No	N/A

Resource	Quota	Adjustable	Maximum quota
Required minimum password characters in password policy	6–99	No	N/A
Email messages sent daily per AWS account[2]	50	No	N/A
Email MFA messages sent to one email address hourly per requester IP address	5-20	No	N/A
Characters in email subject	140	No	N/A
Characters in email message	20,000	No	N/A
Characters in SMS verification message	140	No	N/A
Characters in password	256	No	N/A
Characters in identity provider name	32	No	N/A
Characters in a SAML response	100,000	No	N/A
Identifiers per identity provider	50	No	N/A
Identities linked to a user	5	No	N/A

Resource	Quota	Adjustable	Maximum quota
Passkey/WebAuthn authenticators per user	20	No	N/A
Callback URLs per app client	100	No	N/A
Logout URLs per app client	100	No	N/A
Scopes per resource server	100	No	N/A
Scopes per app client	50	No	N/A
Custom domains per Region	4	No	N/A
Groups to which each user can belong	100	No	N/A
Groups per user pool	10,000	No	N/A

[1] This quota might be encountered in tokens from a Pre token generation Lambda trigger. The number of existing and added claims plus scopes in access and identity tokens in one transaction must add up to a number smaller than or equal to this quota. Suppressed claims and scopes don't contribute to this quota.

[2] This quota applies only if you are using the default email feature for an Amazon Cognito user pool. For a higher email delivery volume, configure your user pool to use your Amazon SES email configuration. This restriction resets daily at 0900 UTC. For more information, see Email settings for Amazon Cognito user pools.

Amazon Cognito user pools session validity parameters

The following quotas describe the available settings for the duration of authentication artifacts and user sessions in user pools.

Token	Quota
ID token	5 minutes – 1 day
Refresh token	1 hour – 3,650 days
Access token	5 minutes – 1 day
Hosted UI session cookie	1 hour
Authentication session token	3 minutes – 15 minutes

Amazon Cognito user pools code security resource quotas (non-adjustable)

The following quotas describe the available time periods related to codes for sign-in, sign-up, and password reset.

Resource	Quota
Sign-up confirmation code validity period	24 hours
User attribute verification code validity period	24 hours
Multi-factor authentication (MFA) code validity period	3–15 minutes
Forgot password code validity period	1 hour
Maximum number of `ConfirmForgotPassword` and `ForgotPassword` requests per user per hour[1]	5–20
Maximum number of `ResendConfirmationCode` requests per user per hour	5
Maximum number of `ConfirmSignUp` requests per user per hour	15

Resource	Quota
Maximum number of `ChangePassword` requests per user per hour	5
Maximum number of `GetUserAttributeVerificationCode` requests per user per hour	5
Maximum number of `VerifyUserAttribute` requests per user per hour	15

[1] Amazon Cognito evaluates risk factors in the request to update passwords and assigns a quota that's tied to the evaluated risk level. For more information, see Forgot password behavior.

Amazon Cognito user pools user import job resource quotas

The following quotas describe the resources and limits available to user import jobs.

Resource	Quota	Adjustable	Maximum quota
User import jobs per user pool	1,000	Yes	Contact your account team.
Maximum characters per user import CSV row	16,000	No	N/A
Maximum CSV file size	100 MB	No	N/A
Maximum number of users per CSV file	500,000	No	N/A

Amazon Cognito identity pools (federated identities) resource quotas

The following quotas describe the maximum number or length of items that you can create in identity pools.

Resource	Quota	Adjustable	Maximum quota
Identity pools per account	1,000	Yes	N/A
Amazon Cognito user pool providers per identity pool	50	Yes	1000
Character length of an identity pool name	128 bytes	No	N/A
Character length of a login provider name	2,048 bytes	No	N/A
Identities per identity pool	Unlimited	No	N/A
Identity providers for which role mappings can be specified	10	No	N/A
Results from a single list or lookup call	60	No	N/A
Role-based access control (RBAC) rules	25	No	N/A

Amazon Cognito Sync resource quotas

The following quotas describe the maximum number or length of items that you can create in Amazon Cognito Sync.

Resource	Quota	Adjustable	Maximum quota
Datasets per identity	20	Yes	Contact your account team.

Resource	Quota	Adjustable	Maximum quota
Records per dataset	1,024	Yes	Contact your account team.
Size of a single dataset	1 MB	Yes	Contact your account team.
Characters in dataset name	128 bytes	No	N/A
Waiting time for a bulk publish after a successful request	24 hours	No	N/A

Multi-Region replication for user pools

With multi-Region replication (MRR), you can create a replica user pool in an additional AWS Region to provide business continuity and disaster recovery capabilities for your authentication infrastructure. With MRR, registered users can continue to authenticate with your applications even when you lose connectivity to resources in a Region, ensuring your applications remain available.

When you configure MRR, Amazon Cognito creates separate user pools with a shared user pool ID. Each replica user pool hosts authentication services for a shared user directory. The primary user pool serves as the authoritative source for administrative configuration and write operations such as password resets and user sign-up. Secondary user pools can't create users. They inherit most settings from the primary user pool and, in a failover state, can handle authentication operations such as user sign-in and token generation.

Things to know about multi-Region replication

- Multi-Region replication has separate add-on costs and requires your user pool to be on the Essentials or Plus feature plan. You can't enable MRR on user pools with the Lite feature plan.
- You must configure your user pool with a multi-Region customer managed key from AWS KMS before enabling replication. The key must be available in all AWS Regions that have user pool replicas. For more information, see the section called “Data encryption”.

- For consistent token validation across Regions, we recommend that you configure your user pool with an updated issuer. For more information, see the section called "OIDC issuer".
- New secondary user pools start in the `INACTIVE` state. Review and configure regional settings before activating the user pool for production use.
- Regional configurations can differ between replicas. You can configure the following settings independently in replicas. All other settings are set in the primary user pool and automatically synchronized to the secondary.
  - Email configuration
  - Email configuration for threat protection notifications
  - SMS configuration
  - Lambda triggers
  - Tags
  - Log export configuration
  - AWS WAF web ACLs
- Data replication between Regions might introduce brief delays. The primary user pool syncs settings and user-directory updates to the secondary, and this process is eventually consistent.

Limitations of multi-Region replication

- You can't generate new users in secondary user pools, either by sign-up or by administrator creation. Federated users can only sign in to a secondary user pool in the failover state if they have previously signed in to the primary user pool.
- Users can't reset their passwords or modify their profiles in secondary user pools. In a failover state, disable these operations in the user interface and make them available after your health check restores access to the primary user pool.
- You can have at most one secondary replica in an additional Region per user directory. Any eligible user pool can have a secondary replica.
- TOTP MFA is not supported in secondary replicas. Users with TOTP MFA configured must authenticate when the user pool in the primary Region is servicing requests.
- The count of password-based authentication attempts before lockout isn't synchronized across Regions. Each replica maintains its own count of failed authentication attempts.

Configuring multi-Region replication

Before you can enable multi-Region replication, ensure your user pool meets the prerequisites: Essentials or Plus feature plan and multi-Region customer managed KMS key.

AWS Management Console

To configure multi-Region replication for a user pool

1. Sign in to the Amazon Cognito console.
2. Choose **User pools**.
3. Choose an existing user pool from the list, or create a new user pool.
4. Choose the **Settings** tab.
5. In the left navigation menu, choose **Multi-Region replication**.
6. Choose **Create a replica user pool**.
7. For **Region**, select the AWS Region where you want to create the replica user pool.
8. Review the configuration summary and choose **Create replica**.
9. After the replica is created, review the regional configuration settings in the comparison table. Configure any Region-specific settings such as email configuration, SMS settings, or Lambda triggers as needed for your replica Region.
10. To configure a Route 53 health check for your domain, navigate to the **Domain services** menu, edit or add a custom domain, and configure a **Route 53 health check ID**.
11. When you're ready to use the replica for production traffic, change the replica status from **Inactive** to **Active**.

API

To create a replica user pool, use the CreateUserPoolReplica operation. The following example creates a replica in the `us-west-2` Region for a primary user pool in `us-east-1`.

```
{
 "UserPoolId": "eusc-de-east-1_EXAMPLE",
 "RegionName": "us-west-2",
 "UserPoolTags": {
    "Environment": "Production",
    "Application": "MyApp"
```

```
 }
}
```

The response includes the replica information:

```
{
 "Replica": {
    "RegionName": "us-west-2",
    "UserPoolArn": "arn:aws:cognito-idp:us-west-2:111122223333:userpool/eusc-de-east-1_EXAMPLE",
    "Status": "PENDING_CREATE",
    "Role": "SECONDARY"
 }
}
```

You must also configure your domain for failover. Set up a health check in Route 53 and apply it to your domain in an UpdateUserPoolDomain request:

```
{
 "CustomDomainConfig": {
    "CertificateArn": "arn:aws:acm:us-east-1:111122223333:certificate/a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
 },
 "Domain": "auth.example.com",
 "ManagedLoginVersion": 2,
 "Routing": {
    "Failover": {
       "SecondaryRegion": "us-west-2",
       "PrimaryRoute53HealthCheckId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
    }
 },
 "UserPoolId": "eusc-de-east-1_EXAMPLE"
}
```

To activate the replica for production use, use the UpdateUserPoolReplica operation:

```
{
 "UserPoolId": "eusc-de-east-1_EXAMPLE",
 "RegionName": "us-west-2",
 "Status": "ACTIVE"
}
```

The response confirms the updated replica status:

```
{
 "Replica": {
    "RegionName": "us-west-2",
    "UserPoolArn": "arn:aws:cognito-idp:us-west-2:111122223333:userpool/eusc-de-east-1_EXAMPLE",
    "Status": "ACTIVE",
    "Role": "SECONDARY"
 }
}
```

Supported API operations in secondary Regions

Amazon Cognito supports a subset of API operations in secondary Regions. The operations available depend on the replica's status. Replicas in `INACTIVE` status support a limited set of read and configuration operations. Replicas in `ACTIVE` status support additional authentication and session management operations. Operations that aren't listed here are only available in the primary Region.

Operations for INACTIVE secondary Regions

Replica user pools in secondary Regions in `INACTIVE` status allow the following Amazon Cognito API operations.

- AdminGetDevice
- AdminGetUser
- AdminListDevices
- AdminListGroupsForUser
- AdminListUserAuthEvents
- AssociateWebACL
- CreateUserPoolDomain
- DeleteUserPoolDomain
- DeleteUserPoolReplica
- DescribeIdentityProvider
- DescribeManagedLoginBranding

- DescribeManagedLoginBrandingByClient
- DescribeResourceServer
- DescribeRiskConfiguration
- DescribeTerms
- DescribeUserPool
- DescribeUserPoolClient
- DescribeUserPoolDomain
- DisassociateWebACL
- GetGroup
- GetLogDeliveryConfiguration
- GetSigningCertificate
- GetUICustomization
- GetUserPoolMfaConfig
- ListIdentityProviders
- ListGroups
- ListResourceServers
- ListTagsForResource
- ListTerms
- ListUserPoolClients
- ListUserPoolClientSecrets
- ListUserPoolReplicas
- ListUserPools
- ListUsers
- ListUsersInGroup
- SetLogDeliveryConfiguration
- SetRiskConfiguration
- SetUserPoolMfaConfig
- TagResource
- UntagResource

- UpdateUserPool
- UpdateUserPoolDomain
- UpdateUserPoolReplica

Additional operations for ACTIVE secondary Regions

Replica user pools in secondary Regions in `ACTIVE` status allow all of the preceding operations, plus the following authentication and session management operations.

- AdminInitiateAuth
- AdminRespondToAuthChallenge
- AdminUpdateAuthEventFeedback
- AdminUserGlobalSignOut
- GetDevice
- GetIdentityProviderByIdentifier
- GetTokensFromRefreshToken
- GetUser
- GetUserAuthFactors
- GlobalSignOut
- InitiateAuth
- ListDevices
- ListWebAuthnCredentials
- RespondToAuthChallenge
- RevokeToken
- UpdateAuthEventFeedback

Failover in multi-Region user pools

With multi-Region user pools, you can fail over managed login, federated login, and direct API calls between two AWS Regions. Managed login and federation failover is available with either a custom domain or a prefix (Cognito) domain configured with your user pool. You can't configure a different custom domain with replica user pools.

Failover for managed login, federation, and machine-to-machine authorization

Failover is available when your primary user pool has a custom domain or a prefix domain. Your user pool domain serves the OAuth 2.0 resources, including the authorize and token endpoints, and handles IdP responses from third-party federation providers, including OIDC, SAML, and social providers.

To enable failover, set up a health check in Route 53 and set the `Routing` field on your domain. You determine what triggers a healthy or unhealthy state. When the health check is in an unhealthy state, Amazon Cognito serves managed login pages and authentication operations from the secondary replica user pool. When the health check enters a healthy state, Amazon Cognito begins routing traffic back to the primary replica.

The DNS record for your custom domain can use Route 53 or any third-party DNS provider. Ensure you have a valid CNAME record in your DNS provider pointing to your target alias, which is a CloudFront distribution. You can find the alias target on the **Domain** page in the Amazon Cognito console.

To update the health check ID in the console

1. Sign in to the Amazon Cognito console.
2. Choose **User pools**, and then choose your user pool.
3. Choose **Domain** under **Branding** from the menu.
4. Under the **Custom domain** section, choose the edit option and select **Edit multi-Region failover**.
5. Toggle the **Enable multi-Region failover** option.
6. Select your Route 53 health check ID from the available health checks.
7. Choose **Save changes**.

Failover for Amazon Cognito APIs and SDKs

If you use the Amazon Cognito APIs or SDKs, there's no usage of a custom domain and your application is responsible for routing traffic to the Amazon Cognito service regional endpoint to handle authentication and other API calls.

If you only have an application frontend using a public client, such as a single-page application (SPA) or mobile app, your application needs to be dynamic to route the API calls accordingly.

Consider a serverless application backend to help determine which Region authentication with Amazon Cognito should begin against.

If you have an application with a backend, the logic to determine which user pool to authenticate against can be determined here.

If you use both managed login endpoints and APIs, use the same Route 53 health check to determine which Region your application directs Amazon Cognito API calls to.

Tagging Amazon Cognito resources

A *tag* is a metadata label that you or AWS assigns to an AWS resource. Each tag consists of a *key* and a *value*. For tags that you assign, you define the key and value. For example, you might define the key as `stage` and the value for one resource as `test`.

Tags help you do the following:

- Identify and organize your AWS resources. Many AWS services support tagging, so that you can assign the same tag to resources from different services. This helps you indicate which resources are related. For example, you could assign the same tag to an Amazon Cognito user pool that you assign to an Amazon DynamoDB table.
- Track your AWS costs. You can activate these tags on the AWS Billing and Cost Management dashboard. AWS uses cost allocation tags to categorize your costs and deliver a monthly cost allocation report to you. For more information, see Use cost allocation tags in the *AWS Billing User Guide*.
- Control access to your resources based on the tags that are assigned to them. You can control access by specifying tag keys and values in the conditions for an AWS Identity and Access Management (IAM) policy. For example, you could allow a user to update a user pool only if the user pool has an `owner` tag with a value of that user's name. For more information, see Controlling access using tags in the *IAM User Guide*.

You can use the AWS Command Line Interface or the Amazon Cognito API to add, edit, or delete tags for both user and identity pools. You can also manage tags for user pools by using the Amazon Cognito console.

For tips on using tags, see the AWS tagging strategies post on the *AWS Answers* blog.

The following sections provide more information about tags for Amazon Cognito.

Supported resources in Amazon Cognito

The following resources in Amazon Cognito support tagging:

- User pools
- Identity pools

Tag restrictions

The following restrictions apply to tags on Amazon Cognito resources:

- Maximum number of tags that you can assign to a resource – 50
- Maximum key length – 128 Unicode characters
- Maximum value length – 256 Unicode characters
- Valid characters for keys and values – a-z, A-Z, 0-9, space, and the following characters: _ . : / = + - @
- Keys and values are case sensitive
- Don't use `aws:` as a prefix for keys; it's reserved for AWS use

Managing tags using the Amazon Cognito console

You can use the Amazon Cognito console to manage the tags that are assigned to your user pools.

To add tags to a user pool

1. Navigate to the Amazon Cognito console. If prompted, enter your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Settings** menu and locate the **Tags** tab.
5. Choose **Add tags** to add your first tag. If you have previously assigned tags to this user pool, in **Manage tags**, chose **Add another**.
6. Specify values for **Tag Key** and **Tag Value**.
7. For each additional tag that you want to add, choose **Add another**.
8. When you are finished adding tags, choose **Save changes**.

To tag an identity pool, navigate to the **Identity pools** menu and select or create an identity pool. In the **Identity pool properties** tab, locate **Tags**. Choose **Add tag**.

AWS CLI examples

The AWS CLI provides commands that help you manage the tags that you assign to your Amazon Cognito user pools and identity pools.

Assigning tags

Use the following commands to assign tags to your existing user pools and identity pools.

Example `tag-resource` Command for user pools

Assign tags to a user pool by using `tag-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp tag-resource \
> --resource-arn user-pool-arn \
> --tags Stage=Test
```

This command includes the following parameters:

- `resource-arn` – The Amazon Resource Name (ARN) of the user pool that you are applying tags to. To look up the ARN, choose the user pool in the Amazon Cognito console, and view the **Pool ARN** value on the **General settings** tab.
- `tags` – The key-value pairs of the tags, in the format *key*=*value*.

To assign multiple tags at once, specify them in a comma-separated list:

```
$ aws cognito-idp tag-resource \
> --resource-arn user-pool-arn \
> --tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Example `tag-resource` Command for identity pools

Assign tags to an identity pool by using `tag-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity tag-resource \
```

```
> --resource-arn identity-pool-arn \
> --tags Stage=Test
```

This command includes the following parameters:

- `resource-arn` – The Amazon Resource Name (ARN) of the identity pool that you are applying tags to. To look up the ARN, choose the identity pool in the Amazon Cognito console, and choose **Edit identity pool**. Then, at **Identity pool ID**, choose **Show ARN**.
- `tags` – The key-value pairs of the tags, in the format *key*=*value*.

To assign multiple tags at once, specify them in a comma-separated list:

```
$ aws cognito-identity tag-resource \
> --resource-arn identity-pool-arn \
> --tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Viewing tags

Use the following commands to view the tags that you have assigned to your user pools and identity pools.

Example `list-tags-for-resource` Command for user pools

View the tags that are assigned to a user pool by using `list-tags-for-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp list-tags-for-resource --resource-arn user-pool-arn
```

Example `list-tags-for-resource` Command for identity pools

View the tags that are assigned to an identity pool by using `list-tags-for-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity list-tags-for-resource --resource-arn identity-pool-arn
```

Removing tags

Use the following commands to remove tags from your user pools and identity pools.

Example `untag-resource` Command for user pools

Remove tags from a user pool by using `untag-resource` within the `cognito-idp` set of commands:

```
$ aws cognito-idp untag-resource \
> --resource-arn user-pool-arn \
> --tag-keys Stage CostCenter Owner
```

For the `--tag-keys` parameter, specify one or more tag keys. Don't include the tag values. Separate keys with spaces.

Example `untag-resource` Command for identity pools

Remove tags from an identity pool by using `untag-resource` within the `cognito-identity` set of commands:

```
$ aws cognito-identity untag-resource \
> --resource-arn identity-pool-arn \
> --tag-keys Stage CostCenter Owner
```

For the `--tag-keys` parameter, specify one or more tag keys. Don't include the tag values.

> **Important**
>
> After you delete a user or identity pool, tags related to the deleted pool can still appear in the console or API calls for up to 30 days after deletion.

Applying tags when you create resources

Use the following commands to assign tags at the moment you create a user pool or identity pool.

Example `create-user-pool` Command with tags

When you create a user pool by using the `create-user-pool` command, you can specify tags with the `--user-pool-tags` parameter:

```
$ aws cognito-idp create-user-pool \
> --pool-name user-pool-name \
```

```
> --user-pool-tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Key-value pairs for tags must be in the format *key*=*value*. If you are adding multiple tags, specify them in a comma-separated list.

Example `create-identity-pool` Command with tags

When you create an identity pool by using the `create-identity-pool` command, you can specify tags with the `--identity-pool-tags` parameter:

```
$ aws cognito-identity create-identity-pool \
> --identity-pool-name identity-pool-name \
> --allow-unauthenticated-identities \
> --identity-pool-tags Stage=Test,CostCenter=80432,Owner=SysEng
```

Key-value pairs for tags must be in the format *key*=*value*. If you are adding multiple tags, specify them in a comma-separated list.

Managing tags using the Amazon Cognito API

You can use the following actions in the Amazon Cognito API to manage the tags for your user pools and identity pools.

API actions for user pool tags

Use the following API actions to assign, view, and remove tags for user pools.

- TagResource
- ListTagsForResource
- UntagResource
- CreateUserPool

API actions for identity pool tags

Use the following API actions to assign, view, and remove tags for identity pools.

- TagResource
- ListTagsForResource

- UntagResource
- CreateIdentityPool

CloudFormation tagging permissions

When you create or update Amazon Cognito user pool or identity pool resources with AWS CloudFormation, CloudFormation can propagate stack-level tags and system-level tags to your resources. To permit this tag propagation, the IAM principal that deploys the stack must have tagging permissions in addition to the resource-creation permissions.

For example, to create a user pool with `cognito-idp:CreateUserPool`, the caller must also have the following permissions:

- `cognito-idp:TagResource`
- `cognito-idp:UntagResource`
- `cognito-idp:ListTagsForResource`

Similarly, to create an identity pool with `cognito-identity:CreateIdentityPool`, the caller must also have the following permissions:

- `cognito-identity:TagResource`
- `cognito-identity:UntagResource`
- `cognito-identity:ListTagsForResource`

Without these permissions, stack creation or update can fail when CloudFormation attempts to apply tags to the resource. If you see an access-denied error related to tagging during a stack operation, verify that your IAM policy includes these permissions.

Troubleshoot Amazon Cognito

This chapter provides solutions to common problems you might encounter while working with Amazon Cognito. Amazon Cognito implementations can face various challenges across authentication flows, user pool configurations, and identity federation setups. Whether you're developing a new application or maintaining an existing one, this troubleshooting guide helps you identify and resolve common issues quickly.

Custom domain configuration errors

When configuring custom domain names in Amazon Cognito, you might receive error messages. Common errors include validation issues, certificate problems, or domain conflicts.

`Custom domain is not a valid subdomain`

This error indicates an issue with DNS resolution for the parent domain. Amazon Cognito does not support top-level domains and requires the parent domain to have a DNS A record for validation.

Problem

This error indicates an issue with DNS resolution for the parent domain. Amazon Cognito does not support top-level domains and requires the parent domain to have a DNS A record for validation.

Solution

- **Create an A Record for the Parent Domain:** You must create an A record in your DNS configuration for the parent domain of your custom domain.
  - **Example:** If your custom domain is `auth.xyz.yourdomain.com`, the parent domain is `xyz.yourdomain.com`. If you want to configure `xyz.yourdomain.com` as a custom domain, the parent is `yourdomain.com`.
  - The parent domain must resolve to a valid IP address. If it doesn't point to a real IP address, you can use a dummy IP address, such as `8.8.8.8`.
- **Verify DNS Propagation (Optional but Recommended):** To ensure your DNS provider has propagated the change, you can run a `dig` command.
  - If using `auth.xyz.yourdomain.com` as the custom domain: `dig A xyz.yourdomain.com +short`
  - If using `xyz.yourdomain.com` as the custom domain: `dig A yourdomain.com +short`
  - The command should return the IP address you configured. If it doesn't, wait until the change has fully propagated.
- **Remove the parent domain A Record:** After successfully creating the custom domain in Amazon Cognito, you can remove the A record you created for the parent domain if it was a dummy one.

For more information, see Using your own domain for the hosted UI.

Domain already associated with another user pool

Custom domain names must be unique across all AWS accounts and Regions.

Problem

Custom domain names must be unique across all AWS accounts and Regions.

Solution

- To use the domain name for a new user pool, you must delete the custom domain from the user pool it's currently associated with.
- **Wait after deletion:** It takes time for the custom domain to fully delete from the first user pool. Creating a new custom domain with the same name immediately after deletion might still result in this error. Wait a few minutes before trying again.

One or more of the CNAMEs that you provided are already associated with a different resource

When you create a custom domain, Amazon Cognito creates an AWS-managed Amazon CloudFront distribution. A domain name can only be used with one Amazon CloudFront distribution. This error occurs if the domain name is already in use as an alternate domain for another Amazon CloudFront distribution.

Problem

When you create a custom domain, Amazon Cognito creates an AWS-managed Amazon CloudFront distribution. A domain name can only be used with one Amazon CloudFront distribution. This error occurs if the domain name is already in use as an alternate domain for another Amazon CloudFront distribution.

Solution

- **Option 1:** Use a different domain name for your Amazon Cognito custom domain.
- **Option 2:** If you use the domain name for Amazon Cognito, do not use it with another Amazon CloudFront distribution.

The specified SSL certificate doesn't exist

Amazon Cognito uses Amazon CloudFront, which requires the AWS Certificate Manager (ACM) certificate to be in the `us-east-1` (N. Virginia) AWS Region, regardless of the user pool's Region.

Problem

Amazon Cognito uses Amazon CloudFront, which requires the AWS Certificate Manager (ACM) certificate to be in the `us-east-1` (N. Virginia) AWS Region, regardless of the user pool's Region.

Solution

- **Ensure Certificate Region:** Confirm the ACM certificate is in the `us-east-1` Region.
- **Check Certificate Validity:** Make sure the selected certificate is not expired.
- **Imported Certificates:** If you imported the certificate into ACM, ensure:
  - It was issued by a public certificate authority.
  - It includes the correct certificate chain.
- **AWS KMS Policy Check:** An explicit `deny` statement in an AWS Key Management Service (AWS KMS) policy for the IAM user or role creating the domain can cause this error. Specifically, check for explicit denials on `kms:DescribeKey`, `kms:CreateGrant`, or `kms:*` actions.

`Invalid refresh token` error

Problem

You receive an `Invalid refresh token` error when trying to use a refresh token to get new access and ID tokens from your Amazon Cognito user pool using the `AdminInitiateAuth` or `InitiateAuth` API operation.

Solution

Implement the following troubleshooting steps based on your user pool configuration and API usage:

- **Ensure consistent app client ID:** When calling the `AdminInitiateAuth` or `InitiateAuth` API for token refresh, you must use the exact same app client ID that was used during the initial authentication that generated the refresh token.
- **Confirm the device:** If you have device tracking enabled on your user pool, but the user's device hasn't been confirmed, you must call the ConfirmDevice API first. After the user confirms their device, you can exchange the refresh token.
- **Include device key in refresh request:** If device tracking is enabled, include the unique device key as an `AuthParameter` when using the `REFRESH_TOKEN_AUTH` flow:

```
{
  "AuthFlow": "REFRESH_TOKEN_AUTH",
  "AuthParameters": {
    "REFRESH_TOKEN": "example_refresh_token",
    "SECRET_HASH": "example_secret_hash", // Required if your app client uses a
 client secret
    "DEVICE_KEY": "example_device_key"
  }
}
```

- **Use USER_SRP_AUTH for device tracking:** If you are using device tracking, the initial authentication flow must be USER_SRP_AUTH.

For more information, see Working with user devices in your user pool.

Invalid SAML response errors in federation

Users receive various `Invalid SAML response` and similar errors when attempting to federate into Amazon Cognito using SAML 2.0. These errors can occur due to attribute mapping issues, certificate problems, or configuration mismatches.

`Invalid user attributes: Required attribute`

Problem

A user is missing a value for an attribute that is required in your user pool, or the IdP is attempting to remove or update an immutable attribute.

Solution

- Check the required attributes defined in your user pool configuration.
- Using network capture tools in your browser, retrieve the SAML response. You might need to perform URL and base64 decoding. Verify the attribute is present in the SAML assertion.
- Sign in to your identity provider and review the attributes that it is sending to Amazon Cognito. Verify that the IdP is configured to send the required attribute using the correct name.
- For immutable attributes, run the following AWS CLI command to identify them: `aws cognito-idp describe-user-pool --user-pool-id USER-POOL-ID --query 'UserPool.SchemaAttributes[?Mutable==`false`].Name'`.

- In the SAML attribute mappings of your IdP, delete any mapping that targets an immutable Amazon Cognito attribute. Alternately, update the destination attribute to a different, mutable attribute.

`Invalid SAML response received: SAML Response signature is invalid`

Problem

Your IdP has updated its SAML signing certificate, causing a mismatch between the certificate in the SAML response and the metadata file stored in Amazon Cognito.

Solution

1. Download the latest metadata file from your IdP.
2. In the Amazon Cognito console, navigate to the **Social and external providers** for your user pool, edit your SAML provider, and replace the existing metadata file with the newly downloaded file.

`Audience restriction` or `Application with identifier not found`

Problem

An incorrect Entity ID is configured in your IdP, or the assertion uses another user pool's Uniform Resource Name (URN).

Solution

1. Get your Amazon Cognito user pool ID from the **Overview** section in the console.
2. In the management console for your IdP, update the entity ID in the SAML application for your user pool. Configure the entity ID to match the format `urn:amazon:cognito:sp:`*`USER_POOL_ID`*. Replace `USER_POOL_ID` with the user pool ID from the previous step.

`An error was encountered with the requested page`

Problem

The Assertion Consumer Service (ACS) URL registered with your IdP is incorrectly configured, or the IdP is not sending the SAML response using the required POST binding.

Solution

- In the management console for your IdP, update the application with the correct ACS URL format, ensuring it uses the `HTTP POST` binding.
- **Default domain format:** `https://cognito-idp.`*`Region`*`.amazonaws.com/`*`your user pool ID`*`/saml2/idpresponse`
- **Custom domain format:** `https://`*`auth.example.com`*`/saml2/idpresponse`

`Invalid relayState from identity provider`

Problem

The `RelayState` parameter is missing or invalid, or the URL is mismatched between the IdP and Amazon Cognito.

Solution

- **For service provider initiated (SP-initiated) flows:** Always start the authentication at your user pool `/oauth2/authorize` endpoint. SAML assertion that don't return `RelayState` parameters from the user's initial visit to Amazon Cognito are not valid SP-initiated requests.
- **For identity provider initiated (IdP-initiated) flows:** The IdP must include the `RelayState` parameter with the SAML assertion to the `/saml2/idpresponse` endpoint, using the required format: `redirect_uri=REDIRECT_URI&state=STATE`.

For more information, see Using SAML identity providers with a user pool.

Managed login users can't select an MFA factor

Problem

Users can't choose their preferred MFA method when signing in through managed login, or they aren't being prompted for the MFA method you expect.

Solution

Amazon Cognito follows specific logic to determine which MFA factors are available to users based on user pool settings, user attributes, and account recovery configuration. For example, users can't use email MFA if email is configured as their primary account recovery method.

To override the default MFA factor selection, you can implement either of these approaches:

- For public applications: Use SetUserMFAPreference with a valid access token to let users set their own MFA preferences
- For administrator-managed applications: Use AdminSetUserMFAPreference with AWS credentials to configure user MFA preferences

Both operations allow you to enable or disable SMS, email, and TOTP MFA methods for individual users, and set one method as preferred.

For more information, see Adding MFA to a user pool.

Passwordless and passkey users can't use MFA

Problem

Users who sign in with one-time password (OTP) authentication methods cannot add or use multi-factor authentication.

Solution

OTP sign-in flows don't support MFA. However, passkey authentication with user verification can satisfy MFA requirements. Set `FactorConfiguration` to `MULTI_FACTOR_WITH_USER_VERIFICATION` in your user pool `WebAuthnConfiguration` to allow passkeys to count as multi-factor authentication.

For more information, see Authentication with user pools.

Not able to receive password reset code through email/SMS

Problem

Users cannot receive verification codes during the forgot password workflow through their email or SMS.

Solution

There are various reasons why verification codes may not be received. Follow this checklist to troubleshoot the issue:

- Check the user's email spam and junk folders.
- Confirm that the user exists in the user pool.
- Verify that the user's status isn't `FORCE_CHANGE_PASSWORD`. If this is the case, the user was created by an administrator and Amazon Cognito will prompt them to set a password when they sign in with their temporary password.
- Check that the user has a verified email or phone number attribute.
- Check your account spend limit for SMS messaging
- Check your Amazon Simple Email Service (Amazon SES) message-sending quota.
- Check if both SMS and Amazon SES (if your user pool is configured to **Send email with Amazon SES**) have been moved out of sandbox. If they were not moved out from sandbox, email addresses or phone numbers which have not been verified by Amazon SES or AWS End User Messaging SMS will not be able to receive password reset codes.
- If the user has an active MFA factor, check that they aren't trying to generate a message to the same factor. For example, users with email MFA active can't sent password reset codes by email.

For more information, see Email settings for Amazon Cognito user pools and SMS message settings for Amazon Cognito user pools.

Password reset fails with unverified recovery attributes: `Could not reset password for the account, please contact support or try again`

Problem

Users receive this error when attempting to reset their password because of unverified recovery methods or MFA configuration conflicts. The error occurs when the user's email or phone number attribute is unverified, or when their MFA settings prevent using the configured recovery method.

Solution

Review your user pool's account recovery configuration and the affected user's verification status:

- **Verify recovery settings:** In your user pool, navigate to **Sign In** > **User Account Recovery**. Ensure **Self service account recovery** is enabled and review your **Recovery message delivery method** setting.
- **Check user attribute verification:** Verify that the user has a verified email address or phone number that matches your recovery method configuration. In the console, go to the user's profile, select **User attributes** > **Edit**, and mark the appropriate attribute as verified.
- **Resolve MFA conflicts:** Users whose preferred MFA method is email can't receive password-reset codes by email, and users with SMS MFA can't receive codes by SMS. Update your **Recovery message delivery method** to provide alternative options such as **SMS if available, otherwise email** or **Email if available, otherwise SMS**.
- **Administrative verification:** Use the API operation AdminUpdateUserAttributes to programmatically verify user attributes when console access isn't available.

For more information, see Passwords, account recovery, and password policies.

SECRET_HASH errors

Problem

Authentication API requests to app clients with client secrets return errors like `An error occurred (NotAuthorizedException) when calling the ForgotPassword operation: Client 1example23456789 is configured with secret but SECRET_HASH was not received.`

Solution

Your application must calculate the `SECRET_HASH` for the current user, app client, and client secret. The calculation method is:

```
Base64 ( HMAC_SHA256 ( "client secret", "Username" + "Client Id" ) )
```

To get the client secret, you can:

1. Open the Amazon Cognito console and navigate to your app client from the **App clients** menu. In the **App client information** section, locate **Client secret**. Select **Show client secret** and your app client secret appears.
2. Generate a DescribeUserPoolClient request. The client secret is included in the response.

For more information, see Computing secret hash values.

The Amazon Cognito console chooses a default configuration for a new user pool

Problem

When you set up a new user pool in the console, Amazon Cognito chooses several default settings for you. Some settings can't be changed after the user pool is created. How can you make informed choices and understand what Amazon Cognito selected automatically?

Solution

The new user pool setup in the Amazon Cognito console is designed for rapid testing and prototyping. The console presents you with only the most critical configuration choices - those that cannot be changed after user pool creation. All other settings that Amazon Cognito configures automatically can be modified later.

We recommend the following approach:

1. Use the console to create test user pools while you refine your implementation.
2. After you've determined your production configuration, apply those settings to your test user pools.
3. Use DescribeUserPool and DescribeUserPoolClient API operations to generate JSON templates of your tested configuration.
4. Use these templates with deployment tools like the AWS SDKs, CDK, REST API, or CloudFormation to create your production resources.

For more information, see Create a new application in the Amazon Cognito console.

Additional troubleshooting resources

For additional troubleshooting guidance and community-contributed solutions, you can also explore the following external resources:

- AWS re:Post Amazon Cognito community - Browse community questions and solutions
- AWS Knowledge Center Amazon Cognito articles - Curated troubleshooting articles

Code examples for Amazon Cognito using AWS SDKs

The following code examples show how to use Amazon Cognito with an AWS software development kit (SDK).

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples

- Code examples for Amazon Cognito Identity using AWS SDKs
  - Basic examples for Amazon Cognito Identity using AWS SDKs
    - Actions for Amazon Cognito Identity using AWS SDKs
      - Use CreateIdentityPool with an AWS SDK or CLI
      - Use DeleteIdentityPool with an AWS SDK or CLI
      - Use DescribeIdentityPool with a CLI
      - Use GetCredentialsForIdentity with an AWS SDK
      - Use GetIdentityPoolRoles with a CLI
      - Use ListIdentityPools with an AWS SDK or CLI
      - Use SetIdentityPoolRoles with a CLI
      - Use UpdateIdentityPool with a CLI
  - Scenarios for Amazon Cognito Identity using AWS SDKs
    - Create an Amazon Textract explorer application
- Code examples for Amazon Cognito Identity Provider using AWS SDKs
  - Basic examples for Amazon Cognito Identity Provider using AWS SDKs
    - Hello Amazon Cognito
    - Actions for Amazon Cognito Identity Provider using AWS SDKs
      - Use AdminCreateUser with an AWS SDK or CLI
      - Use AdminGetUser with an AWS SDK or CLI
      - Use AdminInitiateAuth with an AWS SDK or CLI
      - Use AdminRespondToAuthChallenge with an AWS SDK or CLI
      - Use AdminSetUserPassword with an AWS SDK or CLI

        - Use AssociateSoftwareToken with an AWS SDK or CLI
        - Use ConfirmDevice with an AWS SDK or CLI
        - Use ConfirmForgotPassword with an AWS SDK or CLI
        - Use ConfirmSignUp with an AWS SDK or CLI
        - Use CreateUserPool with an AWS SDK or CLI
        - Use CreateUserPoolClient with an AWS SDK or CLI
        - Use DeleteUser with an AWS SDK or CLI
        - Use DescribeTermsByClient with an AWS SDK
        - Use DescribeUserPoolClient with an AWS SDK or CLI
        - Use ForgotPassword with an AWS SDK or CLI
        - Use GetClientToken with an AWS SDK
        - Use InitiateAuth with an AWS SDK or CLI
        - Use ListUserPools with an AWS SDK or CLI
        - Use ListUsers with an AWS SDK or CLI
        - Use ResendConfirmationCode with an AWS SDK or CLI
        - Use RespondToAuthChallenge with an AWS SDK or CLI
        - Use SignUp with an AWS SDK or CLI
        - Use UpdateUserPool with an AWS SDK or CLI
        - Use VerifySoftwareToken with an AWS SDK or CLI
    - Scenarios for Amazon Cognito Identity Provider using AWS SDKs
        - Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK
        - Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK
        - Get M2M client tokens and discover Terms with Amazon Cognito using an AWS SDK
        - Getting started with user pools
        - Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK
        - Use Amazon Cognito identity pools and authentication flows
        - Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK
- Code examples for Amazon Cognito Sync using AWS SDKs

- Basic examples for Amazon Cognito Sync using AWS SDKs
  - Actions for Amazon Cognito Sync using AWS SDKs
    - Use ListIdentityPoolUsage with an AWS SDK

Code examples for Amazon Cognito Identity using AWS SDKs

The following code examples show how to use Amazon Cognito Identity with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon Cognito Identity

- Basic examples for Amazon Cognito Identity using AWS SDKs
  - Actions for Amazon Cognito Identity using AWS SDKs
    - Use CreateIdentityPool with an AWS SDK or CLI
    - Use DeleteIdentityPool with an AWS SDK or CLI
    - Use DescribeIdentityPool with a CLI
    - Use GetCredentialsForIdentity with an AWS SDK
    - Use GetIdentityPoolRoles with a CLI
    - Use ListIdentityPools with an AWS SDK or CLI
    - Use SetIdentityPoolRoles with a CLI
    - Use UpdateIdentityPool with a CLI
- Scenarios for Amazon Cognito Identity using AWS SDKs
  - Create an Amazon Textract explorer application

Basic examples for Amazon Cognito Identity using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Identity with AWS SDKs.

Examples

- Actions for Amazon Cognito Identity using AWS SDKs
  - Use CreateIdentityPool with an AWS SDK or CLI
  - Use DeleteIdentityPool with an AWS SDK or CLI
  - Use DescribeIdentityPool with a CLI
  - Use GetCredentialsForIdentity with an AWS SDK
  - Use GetIdentityPoolRoles with a CLI
  - Use ListIdentityPools with an AWS SDK or CLI
  - Use SetIdentityPoolRoles with a CLI
  - Use UpdateIdentityPool with a CLI

Actions for Amazon Cognito Identity using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Identity actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Cognito Identity API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon Cognito Identity using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Cognito Identity API Reference.

Examples

- Use CreateIdentityPool with an AWS SDK or CLI
- Use DeleteIdentityPool with an AWS SDK or CLI
- Use DescribeIdentityPool with a CLI
- Use GetCredentialsForIdentity with an AWS SDK
- Use GetIdentityPoolRoles with a CLI

- Use ListIdentityPools with an AWS SDK or CLI
- Use SetIdentityPoolRoles with a CLI
- Use UpdateIdentityPool with a CLI

Use `CreateIdentityPool` with an AWS SDK or CLI

The following code examples show how to use `CreateIdentityPool`.

CLI

AWS CLI

To create an identity pool with Cognito identity pool provider

This example creates an identity pool named MyIdentityPool. It has a Cognito identity pool provider. Unauthenticated identities are not allowed.

Command:

```
aws cognito-identity create-identity-pool --identity-pool-
name MyIdentityPool --no-allow-unauthenticated-identities --cognito-
identity-providers ProviderName="cognito-idp.us-west-2.amazonaws.com/us-
west-2_aaaaaaaaa",ClientId="3n4b5urk1ft4fl3mg5e62d9ado",ServerSideTokenCheck=false
```

Output:

```
{
  "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
  "IdentityPoolName": "MyIdentityPool",
  "AllowUnauthenticatedIdentities": false,
  "CognitoIdentityProviders": [
      {
          "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111111",
          "ClientId": "3n4b5urk1ft4fl3mg5e62d9ado",
          "ServerSideTokenCheck": false
      }
  ]
}
```

- For API details, see CreateIdentityPool in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
 software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolRequest;
import
 software.amazon.awssdk.services.cognitoidentity.model.CreateIdentityPoolResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateIdentityPool {
    public static void main(String[] args) {
        final String usage = """
                Usage:
                    <identityPoolName>\s

                Where:
                    identityPoolName - The name to give your identity pool.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }
```

```
        String identityPoolName = args[0];
        CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
                .region(Region.US_EAST_1)
                .build();

        String identityPoolId = createIdPool(cognitoClient, identityPoolName);
        System.out.println("Unity pool ID " + identityPoolId);
        cognitoClient.close();
    }

    public static String createIdPool(CognitoIdentityClient cognitoClient, String
 identityPoolName) {
        try {
            CreateIdentityPoolRequest poolRequest =
 CreateIdentityPoolRequest.builder()
                    .allowUnauthenticatedIdentities(false)
                    .identityPoolName(identityPoolName)
                    .build();

            CreateIdentityPoolResponse response =
 cognitoClient.createIdentityPool(poolRequest);
            return response.identityPoolId();

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
        return "";
    }
}
```

- For API details, see CreateIdentityPool in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Creates a new Identity Pool which allows unauthenticated identities.

```
New-CGIIdentityPool -AllowUnauthenticatedIdentities $true -IdentityPoolName
 CommonTests13
```

Output:

```
LoggedAt                       : 8/12/2015 4:56:07 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName          :
IdentityPoolId                 : us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3
IdentityPoolName               : CommonTests13
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 136
HttpStatusCode                 : OK
```

- For API details, see CreateIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Creates a new Identity Pool which allows unauthenticated identities.

```
New-CGIIdentityPool -AllowUnauthenticatedIdentities $true -IdentityPoolName
 CommonTests13
```

Output:

```
LoggedAt                       : 8/12/2015 4:56:07 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName          :
IdentityPoolId                 : us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3
IdentityPoolName               : CommonTests13
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 136
HttpStatusCode                 : OK
```

- For API details, see CreateIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSCognitoIdentity


    /// Create a new identity pool and return its ID.
    ///
    /// - Parameters:
    ///     - name: The name to give the new identity pool.
    ///
    /// - Returns: A string containing the newly created pool's ID, or `nil`
    ///   if an error occurred.
    ///
    func createIdentityPool(name: String) async throws -> String? {
        do {
            let cognitoInputCall = CreateIdentityPoolInput(developerProviderName:
 "com.exampleco.CognitoIdentityDemo",
                                                           identityPoolName:
 name)

            let result = try await
 cognitoIdentityClient.createIdentityPool(input: cognitoInputCall)
            guard let poolId = result.identityPoolId else {
                return nil
            }

            return poolId
        } catch {
            print("ERROR: createIdentityPool:", dump(error))
            throw error
        }
    }
```

- For more information, see AWS SDK for Swift developer guide.
- For API details, see CreateIdentityPool in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteIdentityPool` with an AWS SDK or CLI

The following code examples show how to use `DeleteIdentityPool`.

CLI

AWS CLI

To delete identity pool

The following `delete-identity-pool` example deletes the specified identity pool.

Command:

```
aws cognito-identity delete-identity-pool \
    --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111"
```

This command produces no output.

- For API details, see DeleteIdentityPool in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.auth.credentials.ProfileCredentialsProvider;
import software.amazon.awssdk.awscore.exception.AwsServiceException;
import software.amazon.awssdk.regions.Region;
```

```
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
 software.amazon.awssdk.services.cognitoidentity.model.DeleteIdentityPoolRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class DeleteIdentityPool {

    public static void main(String[] args) {
        final String usage = """

                Usage:
                    <identityPoolId>\s

                Where:
                    identityPoolId - The Id value of your identity pool.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String identityPoold = args[0];
        CognitoIdentityClient cognitoIdClient = CognitoIdentityClient.builder()
                .region(Region.US_EAST_1)
                .credentialsProvider(ProfileCredentialsProvider.create())
                .build();

        deleteIdPool(cognitoIdClient, identityPoold);
        cognitoIdClient.close();
    }

    public static void deleteIdPool(CognitoIdentityClient cognitoIdClient, String
 identityPoold) {
        try {
```

```
            DeleteIdentityPoolRequest identityPoolRequest =
 DeleteIdentityPoolRequest.builder()
                    .identityPoolId(identityPoold)
                    .build();

            cognitoIdClient.deleteIdentityPool(identityPoolRequest);
            System.out.println("Done");

        } catch (AwsServiceException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see DeleteIdentityPool in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Deletes a specific Identity Pool.

```
Remove-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

- For API details, see DeleteIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Deletes a specific Identity Pool.

```
Remove-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

- For API details, see DeleteIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSCognitoIdentity


    /// Delete the specified identity pool.
    ///
    /// - Parameters:
    ///   - id: The ID of the identity pool to delete.
    ///
    func deleteIdentityPool(id: String) async throws {
        do {
            let input = DeleteIdentityPoolInput(
                identityPoolId: id
            )

            _ = try await cognitoIdentityClient.deleteIdentityPool(input: input)
        } catch {
            print("ERROR: deleteIdentityPool:", dump(error))
            throw error
        }
    }
```

- For more information, see AWS SDK for Swift developer guide.
- For API details, see DeleteIdentityPool in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeIdentityPool` with a CLI

The following code examples show how to use `DescribeIdentityPool`.

CLI

AWS CLI

To describe an identity pool

This example describes an identity pool.

Command:

```
aws cognito-identity describe-identity-pool --identity-pool-id "us-
west-2:11111111-1111-1111-1111-111111111111"
```

Output:

```
{
  "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
  "IdentityPoolName": "MyIdentityPool",
  "AllowUnauthenticatedIdentities": false,
  "CognitoIdentityProviders": [
      {
          "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111111",
          "ClientId": "3n4b5urk1ft4fl3mg5e62d9ado",
          "ServerSideTokenCheck": false
      }
  ]
}
```

- For API details, see DescribeIdentityPool in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Retrieves information about a specific Identity Pool by its id.

```
Get-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

Output:

```
LoggedAt                       : 8/12/2015 4:29:40 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName          :
IdentityPoolId                 : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName               : CommonTests1
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 142
HttpStatusCode                 : OK
```

- For API details, see DescribeIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Retrieves information about a specific Identity Pool by its id.

```
Get-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

Output:

```
LoggedAt                       : 8/12/2015 4:29:40 PM
AllowUnauthenticatedIdentities : True
DeveloperProviderName          :
IdentityPoolId                 : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName               : CommonTests1
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 142
HttpStatusCode                 : OK
```

- For API details, see DescribeIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetCredentialsForIdentity` with an AWS SDK

The following code example shows how to use `GetCredentialsForIdentity`.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
 software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityRequest;
import
 software.amazon.awssdk.services.cognitoidentity.model.GetCredentialsForIdentityResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class GetIdentityCredentials {
    public static void main(String[] args) {

        final String usage = """

                Usage:
```

```
                    <identityId>\s

            Where:
                identityId - The Id of an existing identity in the format
 REGION:GUID.
            """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String identityId = args[0];
        CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
                .region(Region.US_EAST_1)
                .build();

        getCredsForIdentity(cognitoClient, identityId);
        cognitoClient.close();
    }

    public static void getCredsForIdentity(CognitoIdentityClient cognitoClient,
 String identityId) {
        try {
            GetCredentialsForIdentityRequest getCredentialsForIdentityRequest =
 GetCredentialsForIdentityRequest
                    .builder()
                    .identityId(identityId)
                    .build();

            GetCredentialsForIdentityResponse response = cognitoClient
                    .getCredentialsForIdentity(getCredentialsForIdentityRequest);
            System.out.println(
                    "Identity ID " + response.identityId() + ", Access key ID " +
 response.credentials().accessKeyId());

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see GetCredentialsForIdentity in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetIdentityPoolRoles` with a CLI

The following code examples show how to use `GetIdentityPoolRoles`.

CLI

AWS CLI

To get identity pool roles

This example gets identity pool roles.

Command:

```
aws cognito-identity get-identity-pool-roles --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111"
```

Output:

```
{
  "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
  "Roles": {
      "authenticated": "arn:aws:iam::111111111111:role/Cognito_MyIdentityPoolAuth_Role",
      "unauthenticated": "arn:aws:iam::111111111111:role/Cognito_MyIdentityPoolUnauth_Role"
  }
}
```

- For API details, see GetIdentityPoolRoles in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Gets the information about roles for a specific Identity Pool.

```
Get-CGIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

Output:

```
LoggedAt         : 8/12/2015 4:33:51 PM
IdentityPoolId   : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
Roles            : {[unauthenticated, arn:aws:iam::123456789012:role/
CommonTests1Role]}
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength    : 165
HttpStatusCode   : OK
```

- For API details, see GetIdentityPoolRoles in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Gets the information about roles for a specific Identity Pool.

```
Get-CGIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1
```

Output:

```
LoggedAt         : 8/12/2015 4:33:51 PM
IdentityPoolId   : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
Roles            : {[unauthenticated, arn:aws:iam::123456789012:role/
CommonTests1Role]}
ResponseMetadata : Amazon.Runtime.ResponseMetadata
ContentLength    : 165
HttpStatusCode   : OK
```

- For API details, see GetIdentityPoolRoles in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListIdentityPools` with an AWS SDK or CLI

The following code examples show how to use `ListIdentityPools`.

CLI

AWS CLI

To list identity pools

This example lists identity pools. There s a maximum of 20 identities listed.

Command:

```
aws cognito-identity list-identity-pools --max-results 20
```

Output:

```
{
  "IdentityPools": [
      {
          "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
          "IdentityPoolName": "MyIdentityPool"
      },
      {
          "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
          "IdentityPoolName": "AnotherIdentityPool"
      },
      {
          "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
          "IdentityPoolName": "IdentityPoolRegionA"
      }
  ]
}
```

- For API details, see ListIdentityPools in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import software.amazon.awssdk.services.cognitoidentity.CognitoIdentityClient;
import
 software.amazon.awssdk.services.cognitoidentity.model.ListIdentityPoolsRequest;
import
 software.amazon.awssdk.services.cognitoidentity.model.ListIdentityPoolsResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListIdentityPools {
    public static void main(String[] args) {
        CognitoIdentityClient cognitoClient = CognitoIdentityClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listIdPools(cognitoClient);
        cognitoClient.close();
    }

    public static void listIdPools(CognitoIdentityClient cognitoClient) {
        try {
            ListIdentityPoolsRequest poolsRequest =
 ListIdentityPoolsRequest.builder()
```

```
                    .maxResults(15)
                    .build();

            ListIdentityPoolsResponse response =
 cognitoClient.listIdentityPools(poolsRequest);
            response.identityPools().forEach(pool -> {
                System.out.println("Pool ID: " + pool.identityPoolId());
                System.out.println("Pool name: " + pool.identityPoolName());
            });

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListIdentityPools in *AWS SDK for Java 2.x API Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Retrieves a list of existing Identity Pools.

```
Get-CGIIdentityPoolList
```

Output:

```
IdentityPoolId
 IdentityPoolName
--------------
 ----------------
us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1                    CommonTests1
us-east-1:118d242d-204e-4b88-b803-EXAMPLEGUID2                    Tests2
us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3                    CommonTests13
```

- For API details, see ListIdentityPools in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Retrieves a list of existing Identity Pools.

```
Get-CGIIdentityPoolList
```

Output:

```
IdentityPoolId
 IdentityPoolName
--------------
 ----------------
us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1                    CommonTests1
us-east-1:118d242d-204e-4b88-b803-EXAMPLEGUID2                    Tests2
us-east-1:15d49393-ab16-431a-b26e-EXAMPLEGUID3                    CommonTests13
```

- For API details, see ListIdentityPools in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSCognitoIdentity


    /// Return the ID of the identity pool with the specified name.
    ///
    /// - Parameters:
    ///   - name: The name of the identity pool whose ID should be returned.
    ///
    /// - Returns: A string containing the ID of the specified identity pool
    ///   or `nil` on error or if not found.
    ///
    func getIdentityPoolID(name: String) async throws -> String? {
        let listPoolsInput = ListIdentityPoolsInput(maxResults: 25)
        // Use "Paginated" to get all the objects.
        // This lets the SDK handle the 'nextToken' field in
 "ListIdentityPoolsOutput".
```

```
        let pages = cognitoIdentityClient.listIdentityPoolsPaginated(input:
 listPoolsInput)

        do {
            for try await page in pages {
                guard let identityPools = page.identityPools else {
                    print("ERROR: listIdentityPoolsPaginated returned nil
 contents.")
                    continue
                }

                /// Read pages of identity pools from Cognito until one is found
                /// whose name matches the one specified in the `name` parameter.
                /// Return the matching pool's ID.

                for pool in identityPools {
                    if pool.identityPoolName == name {
                        return pool.identityPoolId!
                    }
                }
            }
        } catch {
            print("ERROR: getIdentityPoolID:", dump(error))
            throw error
        }

        return nil
    }
```

Get the ID of an existing identity pool or create it if it doesn't already exist.

```
import AWSCognitoIdentity


    /// Return the ID of the identity pool with the specified name.
    ///
    /// - Parameters:
    ///   - name: The name of the identity pool whose ID should be returned
    ///
    /// - Returns: A string containing the ID of the specified identity pool.
    ///   Returns `nil` if there's an error or if the pool isn't found.
```

```
    ///
    public func getOrCreateIdentityPoolID(name: String) async throws -> String? {
        // See if the pool already exists. If it doesn't, create it.

        do {
            guard let poolId = try await getIdentityPoolID(name: name) else {
                return try await createIdentityPool(name: name)
            }

            return poolId
        } catch {
            print("ERROR: getOrCreateIdentityPoolID:", dump(error))
            throw error
        }
    }
```

- For more information, see AWS SDK for Swift developer guide.
- For API details, see ListIdentityPools in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SetIdentityPoolRoles` with a CLI

The following code examples show how to use `SetIdentityPoolRoles`.

CLI

AWS CLI

To set identity pool roles

The following `set-identity-pool-roles` example sets an identity pool role.

```
aws cognito-identity set-identity-pool-roles \
    --identity-pool-id "us-west-2:11111111-1111-1111-1111-111111111111" \
    --roles authenticated="arn:aws:iam::111111111111:role/
Cognito_MyIdentityPoolAuth_Role"
```

- For API details, see SetIdentityPoolRoles in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Configures the specific Identity Pool to have an unauthenticated IAM role.

```
Set-CGIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1 -Role @{ "unauthenticated" = "arn:aws:iam::123456789012:role/
CommonTests1Role" }
```

- For API details, see SetIdentityPoolRoles in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Configures the specific Identity Pool to have an unauthenticated IAM role.

```
Set-CGIIdentityPoolRole -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1 -Role @{ "unauthenticated" = "arn:aws:iam::123456789012:role/
CommonTests1Role" }
```

- For API details, see SetIdentityPoolRoles in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateIdentityPool` with a CLI

The following code examples show how to use `UpdateIdentityPool`.

CLI

AWS CLI

To update an identity pool

This example updates an identity pool. It sets the name to MyIdentityPool. It adds Cognito as an identity provider. It disallows unauthenticated identities.

Command:

```
aws cognito-identity update-identity-pool --identity-pool-id "us-
west-2:11111111-1111-1111-1111-111111111111" --identity-pool-
name "MyIdentityPool" --no-allow-unauthenticated-identities --cognito-
identity-providers ProviderName="cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111111",ClientId="3n4b5urk1ft4fl3mg5e62d9ado",ServerSideTokenCheck=false
```

Output:

```
{
  "IdentityPoolId": "us-west-2:11111111-1111-1111-1111-111111111111",
  "IdentityPoolName": "MyIdentityPool",
  "AllowUnauthenticatedIdentities": false,
  "CognitoIdentityProviders": [
      {
          "ProviderName": "cognito-idp.us-west-2.amazonaws.com/us-
west-2_111111111",
          "ClientId": "3n4b5urk1ft4fl3mg5e62d9ado",
          "ServerSideTokenCheck": false
      }
  ]
}
```

- For API details, see UpdateIdentityPool in *AWS CLI Command Reference*.

PowerShell

Tools for PowerShell V4

Example 1: Updates some of the Identity Pool properties, in this case the name of the Identity Pool.

```
Update-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1 -IdentityPoolName NewPoolName
```

Output:

```
LoggedAt                      : 8/12/2015 4:53:33 PM
AllowUnauthenticatedIdentities : False
DeveloperProviderName         :
```

```
IdentityPoolId                 : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName               : NewPoolName
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 135
HttpStatusCode                 : OK
```

- For API details, see UpdateIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V4)*.

Tools for PowerShell V5

Example 1: Updates some of the Identity Pool properties, in this case the name of the Identity Pool.

```
Update-CGIIdentityPool -IdentityPoolId us-east-1:0de2af35-2988-4d0b-b22d-
EXAMPLEGUID1 -IdentityPoolName NewPoolName
```

Output:

```
LoggedAt                       : 8/12/2015 4:53:33 PM
AllowUnauthenticatedIdentities : False
DeveloperProviderName          :
IdentityPoolId                 : us-east-1:0de2af35-2988-4d0b-b22d-EXAMPLEGUID1
IdentityPoolName               : NewPoolName
OpenIdConnectProviderARNs      : {}
SupportedLoginProviders        : {}
ResponseMetadata               : Amazon.Runtime.ResponseMetadata
ContentLength                  : 135
HttpStatusCode                 : OK
```

- For API details, see UpdateIdentityPool in *AWS Tools for PowerShell Cmdlet Reference (V5)*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Cognito Identity using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Cognito Identity with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling

multiple functions within Amazon Cognito Identity or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Create an Amazon Textract explorer application

Create an Amazon Textract explorer application

The following code examples show how to explore Amazon Textract output through an interactive application.

JavaScript

SDK for JavaScript (v3)

Shows how to use the AWS SDK for JavaScript to build a React application that uses Amazon Textract to extract data from a document image and display it in an interactive web page. This example runs in a web browser and requires an authenticated Amazon Cognito identity for credentials. It uses Amazon Simple Storage Service (Amazon S3) for storage, and for notifications it polls an Amazon Simple Queue Service (Amazon SQS) queue that is subscribed to an Amazon Simple Notification Service (Amazon SNS) topic.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

Python

SDK for Python (Boto3)

Shows how to use the AWS SDK for Python (Boto3) with Amazon Textract to detect text, form, and table elements in a document image. The input image and Amazon Textract output are shown in a Tkinter application that lets you explore the detected elements.

- Submit a document image to Amazon Textract and explore the output of detected elements.
- Submit images directly to Amazon Textract or through an Amazon Simple Storage Service (Amazon S3) bucket.
- Use asynchronous APIs to start a job that publishes a notification to an Amazon Simple Notification Service (Amazon SNS) topic when the job completes.
- Poll an Amazon Simple Queue Service (Amazon SQS) queue for a job completion message and display the results.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity
- Amazon S3
- Amazon SNS
- Amazon SQS
- Amazon Textract

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show how to use Amazon Cognito Identity Provider with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

Scenarios are code examples that show you how to accomplish specific tasks by calling multiple functions within a service or combined with other AWS services.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon Cognito Identity Provider

- Basic examples for Amazon Cognito Identity Provider using AWS SDKs
  - Hello Amazon Cognito
  - Actions for Amazon Cognito Identity Provider using AWS SDKs
    - Use AdminCreateUser with an AWS SDK or CLI
    - Use AdminGetUser with an AWS SDK or CLI
    - Use AdminInitiateAuth with an AWS SDK or CLI
    - Use AdminRespondToAuthChallenge with an AWS SDK or CLI
    - Use AdminSetUserPassword with an AWS SDK or CLI
    - Use AssociateSoftwareToken with an AWS SDK or CLI
    - Use ConfirmDevice with an AWS SDK or CLI
    - Use ConfirmForgotPassword with an AWS SDK or CLI
    - Use ConfirmSignUp with an AWS SDK or CLI
    - Use CreateUserPool with an AWS SDK or CLI
    - Use CreateUserPoolClient with an AWS SDK or CLI
    - Use DeleteUser with an AWS SDK or CLI
    - Use DescribeTermsByClient with an AWS SDK
    - Use DescribeUserPoolClient with an AWS SDK or CLI
    - Use ForgotPassword with an AWS SDK or CLI
    - Use GetClientToken with an AWS SDK
    - Use InitiateAuth with an AWS SDK or CLI
    - Use ListUserPools with an AWS SDK or CLI
    - Use ListUsers with an AWS SDK or CLI

    - Use ResendConfirmationCode with an AWS SDK or CLI
    - Use RespondToAuthChallenge with an AWS SDK or CLI
    - Use SignUp with an AWS SDK or CLI
    - Use UpdateUserPool with an AWS SDK or CLI
    - Use VerifySoftwareToken with an AWS SDK or CLI
- Scenarios for Amazon Cognito Identity Provider using AWS SDKs
    - Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK
    - Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK
    - Get M2M client tokens and discover Terms with Amazon Cognito using an AWS SDK
    - Getting started with user pools
    - Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK
    - Use Amazon Cognito identity pools and authentication flows
    - Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK

Basic examples for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Identity Provider with AWS SDKs.

Examples

- Hello Amazon Cognito
- Actions for Amazon Cognito Identity Provider using AWS SDKs
    - Use AdminCreateUser with an AWS SDK or CLI
    - Use AdminGetUser with an AWS SDK or CLI
    - Use AdminInitiateAuth with an AWS SDK or CLI
    - Use AdminRespondToAuthChallenge with an AWS SDK or CLI
    - Use AdminSetUserPassword with an AWS SDK or CLI
    - Use AssociateSoftwareToken with an AWS SDK or CLI
    - Use ConfirmDevice with an AWS SDK or CLI

- Use ConfirmForgotPassword with an AWS SDK or CLI
- Use ConfirmSignUp with an AWS SDK or CLI
- Use CreateUserPool with an AWS SDK or CLI
- Use CreateUserPoolClient with an AWS SDK or CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DescribeTermsByClient with an AWS SDK
- Use DescribeUserPoolClient with an AWS SDK or CLI
- Use ForgotPassword with an AWS SDK or CLI
- Use GetClientToken with an AWS SDK
- Use InitiateAuth with an AWS SDK or CLI
- Use ListUserPools with an AWS SDK or CLI
- Use ListUsers with an AWS SDK or CLI
- Use ResendConfirmationCode with an AWS SDK or CLI
- Use RespondToAuthChallenge with an AWS SDK or CLI
- Use SignUp with an AWS SDK or CLI
- Use UpdateUserPool with an AWS SDK or CLI
- Use VerifySoftwareToken with an AWS SDK or CLI

Hello Amazon Cognito

The following code examples show how to get started using Amazon Cognito.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Code for the CMakeLists.txt CMake file.

```
# Set the minimum required version of CMake for this project.
```

```
cmake_minimum_required(VERSION 3.13)

# Set the AWS service components used by this project.
set(SERVICE_COMPONENTS cognito-idp)

# Set this project's name.
project("hello_cognito")

# Set the C++ standard to use to build this target.
# At least C++ 11 is required for the AWS SDK for C++.
set(CMAKE_CXX_STANDARD 11)

# Use the MSVC variable to determine if this is a Windows build.
set(WINDOWS_BUILD ${MSVC})

if (WINDOWS_BUILD) # Set the location where CMake can find the installed
 libraries for the AWS SDK.
    string(REPLACE ";" "/aws-cpp-sdk-all;" SYSTEM_MODULE_PATH
 "${CMAKE_SYSTEM_PREFIX_PATH}/aws-cpp-sdk-all")
    list(APPEND CMAKE_PREFIX_PATH ${SYSTEM_MODULE_PATH})
endif ()

# Find the AWS SDK for C++ package.
find_package(AWSSDK REQUIRED COMPONENTS ${SERVICE_COMPONENTS})

if (WINDOWS_BUILD AND AWSSDK_INSTALL_AS_SHARED_LIBS)
     # Copy relevant AWS SDK for C++ libraries into the current binary directory
 for running and debugging.

     # set(BIN_SUB_DIR "/Debug") # If you are building from the command line, you
 may need to uncomment this
                                     # and set the proper subdirectory to the
 executables' location.

     AWSSDK_CPY_DYN_LIBS(SERVICE_COMPONENTS ""
 ${CMAKE_CURRENT_BINARY_DIR}${BIN_SUB_DIR})
endif ()

add_executable(${PROJECT_NAME}
        hello_cognito.cpp)

target_link_libraries(${PROJECT_NAME}
        ${AWSSDK_LINK_LIBRARIES})
```

Code for the hello_cognito.cpp source file.

```
#include <aws/core/Aws.h>
#include <aws/cognito-idp/CognitoIdentityProviderClient.h>
#include <aws/cognito-idp/model/ListUserPoolsRequest.h>
#include <iostream>

/*
 *  A "Hello Cognito" starter application which initializes an Amazon Cognito
 client and lists the Amazon Cognito
 *  user pools.
 *
 *  main function
 *
 *  Usage: 'hello_cognito'
 *
 */

int main(int argc, char **argv) {
    Aws::SDKOptions options;
    // Optionally change the log level for debugging.
//   options.loggingOptions.logLevel = Utils::Logging::LogLevel::Debug;
    Aws::InitAPI(options); // Should only be called once.
    int result = 0;
    {
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

        Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
 cognitoClient(clientConfig);

        Aws::String nextToken; // Used for pagination.
        std::vector<Aws::String> userPools;

        do {
            Aws::CognitoIdentityProvider::Model::ListUserPoolsRequest
 listUserPoolsRequest;
            if (!nextToken.empty()) {
                listUserPoolsRequest.SetNextToken(nextToken);
            }
```

```
            Aws::CognitoIdentityProvider::Model::ListUserPoolsOutcome
 listUserPoolsOutcome =
                    cognitoClient.ListUserPools(listUserPoolsRequest);

            if (listUserPoolsOutcome.IsSuccess()) {
                for (auto &userPool:
 listUserPoolsOutcome.GetResult().GetUserPools()) {

                    userPools.push_back(userPool.GetName());
                }

                nextToken = listUserPoolsOutcome.GetResult().GetNextToken();
            } else {
                std::cerr << "ListUserPools error: " <<
 listUserPoolsOutcome.GetError().GetMessage() << std::endl;
                result = 1;
                break;
            }


        } while (!nextToken.empty());
        std::cout << userPools.size() << " user pools found." << std::endl;
        for (auto &userPool: userPools) {
            std::cout << "   user pool: " << userPool << std::endl;
        }
    }

    Aws::ShutdownAPI(options); // Should only be called once.
    return result;
}
```

- For API details, see ListUserPools in *AWS SDK for C++ API Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
 Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the user pools for your account.")
 var pools []types.UserPoolDescriptionType
 paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
```

```
  cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:
 aws.Int32(10)})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get user pools. Here's why: %v\n", err)
  } else {
   pools = append(pools, output.UserPools...)
  }
 }
 if len(pools) == 0 {
  fmt.Println("You don't have any user pools!")
 } else {
  for _, pool := range pools {
   fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)
  }
 }
}
```

- For API details, see ListUserPools in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;
```

```
/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListUserPools {
    public static void main(String[] args) {
        CognitoIdentityProviderClient cognitoClient =
 CognitoIdentityProviderClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listAllUserPools(cognitoClient);
        cognitoClient.close();
    }

    public static void listAllUserPools(CognitoIdentityProviderClient
 cognitoClient) {
        try {
            ListUserPoolsRequest request = ListUserPoolsRequest.builder()
                    .maxResults(10)
                    .build();

            ListUserPoolsResponse response =
 cognitoClient.listUserPools(request);
            response.userPools().forEach(userpool -> {
                System.out.println("User pool " + userpool.name() + ", User ID "
 + userpool.id());
            });

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListUserPools in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import {
  paginateListUserPools,
  CognitoIdentityProviderClient,
} from "@aws-sdk/client-cognito-identity-provider";

const client = new CognitoIdentityProviderClient({});

export const helloCognito = async () => {
  const paginator = paginateListUserPools({ client }, {});

  const userPoolNames = [];

  for await (const page of paginator) {
    const names = page.UserPools.map((pool) => pool.Name);
    userPoolNames.push(...names);
  }

  console.log("User pool names: ");
  console.log(userPoolNames.join("\n"));
  return userPoolNames;
};
```

- For API details, see ListUserPools in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import boto3

# Create a Cognito Identity Provider client
cognitoidp = boto3.client("cognito-idp")

# Initialize a paginator for the list_user_pools operation
paginator = cognitoidp.get_paginator("list_user_pools")

# Create a PageIterator from the paginator
page_iterator = paginator.paginate(MaxResults=10)

# Initialize variables for pagination
user_pools = []

# Handle pagination
for page in page_iterator:
    user_pools.extend(page.get("UserPools", []))

# Print the list of user pools
print("User Pools for the account:")
if user_pools:
    for pool in user_pools:
        print(f"Name: {pool['Name']}, ID: {pool['Id']}")
else:
    print("No user pools found.")
```

- For API details, see ListUserPools in *AWS SDK for Python (Boto3) API Reference*.

Ruby

SDK for Ruby

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
require 'aws-sdk-cognitoidentityprovider'
require 'logger'

# CognitoManager is a class responsible for managing AWS Cognito operations
# such as listing all user pools in the current AWS account.
class CognitoManager
  def initialize(client)
    @client = client
    @logger = Logger.new($stdout)
  end

  # Lists and prints all user pools associated with the AWS account.
  def list_user_pools
    paginator = @client.list_user_pools(max_results: 10)
    user_pools = []
    paginator.each_page do |page|
      user_pools.concat(page.user_pools)
    end

    if user_pools.empty?
      @logger.info('No Cognito user pools found.')
    else
      user_pools.each do |user_pool|
        @logger.info("User pool ID: #{user_pool.id}")
        @logger.info("User pool name: #{user_pool.name}")
        @logger.info("User pool status: #{user_pool.status}")
        @logger.info('---')
      end
    end
  end
end
```

```
if $PROGRAM_NAME == __FILE__
  cognito_client = Aws::CognitoIdentityProvider::Client.new
  manager = CognitoManager.new(cognito_client)
  manager.list_user_pools
end
```

- For API details, see ListUserPools in *AWS SDK for Ruby API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Actions for Amazon Cognito Identity Provider using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Identity Provider actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

These excerpts call the Amazon Cognito Identity Provider API and are code excerpts from larger programs that must be run in context. You can see actions in context in Scenarios for Amazon Cognito Identity Provider using AWS SDKs.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Cognito Identity Provider API Reference.

Examples

- Use AdminCreateUser with an AWS SDK or CLI
- Use AdminGetUser with an AWS SDK or CLI
- Use AdminInitiateAuth with an AWS SDK or CLI
- Use AdminRespondToAuthChallenge with an AWS SDK or CLI
- Use AdminSetUserPassword with an AWS SDK or CLI
- Use AssociateSoftwareToken with an AWS SDK or CLI
- Use ConfirmDevice with an AWS SDK or CLI
- Use ConfirmForgotPassword with an AWS SDK or CLI

- Use ConfirmSignUp with an AWS SDK or CLI
- Use CreateUserPool with an AWS SDK or CLI
- Use CreateUserPoolClient with an AWS SDK or CLI
- Use DeleteUser with an AWS SDK or CLI
- Use DescribeTermsByClient with an AWS SDK
- Use DescribeUserPoolClient with an AWS SDK or CLI
- Use ForgotPassword with an AWS SDK or CLI
- Use GetClientToken with an AWS SDK
- Use InitiateAuth with an AWS SDK or CLI
- Use ListUserPools with an AWS SDK or CLI
- Use ListUsers with an AWS SDK or CLI
- Use ResendConfirmationCode with an AWS SDK or CLI
- Use RespondToAuthChallenge with an AWS SDK or CLI
- Use SignUp with an AWS SDK or CLI
- Use UpdateUserPool with an AWS SDK or CLI
- Use VerifySoftwareToken with an AWS SDK or CLI

Use `AdminCreateUser` with an AWS SDK or CLI

The following code examples show how to use `AdminCreateUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Getting started with user pools
- Write custom activity data with a Lambda function after Amazon Cognito user authentication

CLI

AWS CLI

To create a user

The following `admin-create-user` example creates a user with the specified settings email address and phone number.

```
aws cognito-idp admin-create-user \
    --user-pool-id us-west-2_aaaaaaaaa \
    --username diego \
    --user-attributes Name=email,Value=diego@example.com
 Name=phone_number,Value="+15555551212" \
    --message-action SUPPRESS
```

Output:

```
{
    "User": {
        "Username": "diego",
        "Attributes": [
            {
                "Name": "sub",
                "Value": "7325c1de-b05b-4f84-b321-9adc6e61f4a2"
            },
            {
                "Name": "phone_number",
                "Value": "+15555551212"
            },
            {
                "Name": "email",
                "Value": "diego@example.com"
            }
        ],
        "UserCreateDate": 1548099495.428,
        "UserLastModifiedDate": 1548099495.428,
        "Enabled": true,
        "UserStatus": "FORCE_CHANGE_PASSWORD"
    }
}
```

- For API details, see AdminCreateUser in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
```

```
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

- For API details, see AdminCreateUser in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AdminGetUser` with an AWS SDK or CLI

The following code examples show how to use `AdminGetUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with
administrator access.
```

```
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }
```

- For API details, see AdminGetUser in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

    Aws::CognitoIdentityProvider::Model::AdminGetUserRequest request;
    request.SetUsername(userName);
```

```
    request.SetUserPoolId(userPoolID);

    Aws::CognitoIdentityProvider::Model::AdminGetUserOutcome outcome =
            client.AdminGetUser(request);

    if (outcome.IsSuccess()) {
        std::cout << "The status for " << userName << " is " <<

 Aws::CognitoIdentityProvider::Model::UserStatusTypeMapper::GetNameForUserStatusType(
                          outcome.GetResult().GetUserStatus()) << std::endl;
        std::cout << "Enabled is " << outcome.GetResult().GetEnabled() <<
 std::endl;
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::AdminGetUser. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }
```

- For API details, see AdminGetUser in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To get a user

This example gets information about username jane@example.com.

Command:

```
aws cognito-idp admin-get-user --user-pool-id us-west-2_aaaaaaaaa --
username jane@example.com
```

Output:

```
{
  "Username": "4320de44-2322-4620-999b-5e2e1c8df013",
  "Enabled": true,
  "UserStatus": "FORCE_CHANGE_PASSWORD",
  "UserCreateDate": 1548108509.537,
```

```
  "UserAttributes": [
      {
          "Name": "sub",
          "Value": "4320de44-2322-4620-999b-5e2e1c8df013"
      },
      {
          "Name": "email_verified",
          "Value": "true"
      },
      {
          "Name": "phone_number_verified",
          "Value": "true"
      },
      {
          "Name": "phone_number",
          "Value": "+01115551212"
      },
      {
          "Name": "email",
          "Value": "jane@example.com"
      }
  ],
  "UserLastModifiedDate": 1548108509.537
}
```

- For API details, see AdminGetUser in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void getAdminUser(CognitoIdentityProviderClient
identityProviderClient, String userName,
           String poolId) {
       try {
           AdminGetUserRequest userRequest = AdminGetUserRequest.builder()
```

```
                    .username(userName)
                    .userPoolId(poolId)
                    .build();

            AdminGetUserResponse response =
 identityProviderClient.adminGetUser(userRequest);
            System.out.println("User status " + response.userStatusAsString());

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see AdminGetUser in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const adminGetUser = ({ userPoolId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminGetUserCommand({
    UserPoolId: userPoolId,
    Username: username,
  });

  return client.send(command);
};
```

- For API details, see AdminGetUser in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getAdminUser(
    userNameVal: String?,
    poolIdVal: String?,
) {
    val userRequest =
        AdminGetUserRequest {
            username = userNameVal
            userPoolId = poolIdVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.adminGetUser(userRequest)
        println("User status ${response.userStatus}")
    }
}
```

- For API details, see AdminGetUser in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def sign_up_user(self, user_name, password, user_email):
        """
        Signs up a new user with Amazon Cognito. This action prompts Amazon
 Cognito
        to send an email to the specified email address. The email contains a
 code that
        can be used to confirm the user.

        When the user already exists, the user status is checked to determine
 whether
        the user has been confirmed.

        :param user_name: The user name that identifies the new user.
        :param password: The password for the new user.
        :param user_email: The email address for the new user.
        :return: True when the user is already confirmed with Amazon Cognito.
                 Otherwise, false.
        """
        try:
            kwargs = {
                "ClientId": self.client_id,
                "Username": user_name,
                "Password": password,
                "UserAttributes": [{"Name": "email", "Value": user_email}],
```

```
            }
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            response = self.cognito_idp_client.sign_up(**kwargs)
            confirmed = response["UserConfirmed"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "UsernameExistsException":
                response = self.cognito_idp_client.admin_get_user(
                    UserPoolId=self.user_pool_id, Username=user_name
                )
                logger.warning(
                    "User %s exists and is %s.", user_name,
 response["UserStatus"]
                )
                confirmed = response["UserStatus"] == "CONFIRMED"
            else:
                logger.error(
                    "Couldn't sign up %s. Here's why: %s: %s",
                    user_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return confirmed
```

- For API details, see AdminGetUser in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider
```

```
    /// Get information about a specific user in a user pool.
    ///
    /// - Parameters:
    ///   - cipClient: The Amazon Cognito Identity Provider client to use.
    ///   - userName: The user to retrieve information about.
    ///   - userPoolId: The user pool to search for the specified user.
    ///
    /// - Returns: `true` if the user's information was successfully
    ///   retrieved. Otherwise returns `false`.
    func adminGetUser(cipClient: CognitoIdentityProviderClient, userName: String,
                      userPoolId: String) async -> Bool {
        do {
            let output = try await cipClient.adminGetUser(
                input: AdminGetUserInput(
                    userPoolId: userPoolId,
                    username: userName
                )
            )

            guard let userStatus = output.userStatus else {
                print("*** Unable to get the user's status.")
                return false
            }

            print("User status: \(userStatus)")
            return true
        } catch {
            return false
        }
    }
```

- For API details, see AdminGetUser in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AdminInitiateAuth` with an AWS SDK or CLI

The following code examples show how to use `AdminInitiateAuth`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }
```

- For API details, see AdminInitiateAuth in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthRequest request;
    request.SetClientId(clientID);
    request.SetUserPoolId(userPoolID);
    request.AddAuthParameters("USERNAME", userName);
    request.AddAuthParameters("PASSWORD", password);
    request.SetAuthFlow(

Aws::CognitoIdentityProvider::Model::AuthFlowType::ADMIN_USER_PASSWORD_AUTH);


    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthOutcome outcome =
            client.AdminInitiateAuth(request);

    if (outcome.IsSuccess()) {
        std::cout << "Call to AdminInitiateAuth was successful." << std::endl;
        sessionResult = outcome.GetResult().GetSession();
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::AdminInitiateAuth. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
```

```
    }
```

- For API details, see AdminInitiateAuth in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To sign in a user as an admin

The following `admin-initiate-auth` example signs in the user diego@example.com. This example also includes metadata for threat protection and ClientMetadata for Lambda triggers. The user is configured for TOTP MFA and receives a challenge to provide a code from their authenticator app before they can complete authentication.

```
aws cognito-idp admin-initiate-auth \
    --user-pool-id us-west-2_EXAMPLE \
    --client-id 1example23456789 \
    --auth-flow ADMIN_USER_PASSWORD_AUTH \
    --auth-parameters USERNAME=diego@example.com,PASSWORD="My@Example
$Password3!",SECRET_HASH=ExampleEncodedClientIdSecretAndUsername= \
    --context-data="{\"EncodedData\":\"abc123example\",\"HttpHeaders\":
[{\"headerName\":\"UserAgent\",\"headerValue\":\"Mozilla/5.0 (Windows NT
 6.1; Win64; x64; rv:47.0) Gecko/20100101 Firefox/47.0\"}],\"IpAddress\":
\"192.0.2.1\",\"ServerName\":\"example.com\",\"ServerPath\":\"/login\"}" \
    --client-metadata="{\"MyExampleKey\": \"MyExampleValue\"}"
```

Output:

```
{
    "ChallengeName": "SOFTWARE_TOKEN_MFA",
    "Session": "AYABeExample...",
    "ChallengeParameters": {
        "FRIENDLY_DEVICE_NAME": "MyAuthenticatorApp",
        "USER_ID_FOR_SRP": "diego@example.com"
    }
}
```

For more information, see Admin authentication flow in the *Amazon Cognito Developer Guide*.

- For API details, see AdminInitiateAuth in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static AdminInitiateAuthResponse
initiateAuth(CognitoIdentityProviderClient identityProviderClient,
            String clientId, String userName, String password, String userPoolId)
{
       try {
           Map<String, String> authParameters = new HashMap<>();
           authParameters.put("USERNAME", userName);
           authParameters.put("PASSWORD", password);

           AdminInitiateAuthRequest authRequest =
AdminInitiateAuthRequest.builder()
                   .clientId(clientId)
                   .userPoolId(userPoolId)
                   .authParameters(authParameters)
                   .authFlow(AuthFlowType.ADMIN_USER_PASSWORD_AUTH)
                   .build();

           AdminInitiateAuthResponse response =
identityProviderClient.adminInitiateAuth(authRequest);
           System.out.println("Result Challenge is : " +
response.challengeName());
           return response;

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }

       return null;
   }
```

- For API details, see AdminInitiateAuth in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};
```

- For API details, see AdminInitiateAuth in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun checkAuthMethod(
    clientIdVal: String,
    userNameVal: String,
    passwordVal: String,
    userPoolIdVal: String,
): AdminInitiateAuthResponse {
    val authParas = mutableMapOf<String, String>()
    authParas["USERNAME"] = userNameVal
    authParas["PASSWORD"] = passwordVal

    val authRequest =
        AdminInitiateAuthRequest {
            clientId = clientIdVal
            userPoolId = userPoolIdVal
            authParameters = authParas
            authFlow = AuthFlowType.AdminUserPasswordAuth
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.adminInitiateAuth(authRequest)
        println("Result Challenge is ${response.challengeName}")
        return response
    }
}
```

- For API details, see AdminInitiateAuth in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""
```

```
    def __init__(self, cognito_idp_client, user_pool_id, client_id,
client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def start_sign_in(self, user_name, password):
        """
        Starts the sign-in process for a user by using administrator credentials.
        This method of signing in is appropriate for code running on a secure
server.

        If the user pool is configured to require MFA and this is the first sign-
in
        for the user, Amazon Cognito returns a challenge response to set up an
        MFA application. When this occurs, this function gets an MFA secret from
        Amazon Cognito and returns it to the caller.

        :param user_name: The name of the user to sign in.
        :param password: The user's password.
        :return: The result of the sign-in attempt. When sign-in is successful,
this
                 returns an access token that can be used to get AWS credentials.
Otherwise,
                 Amazon Cognito returns a challenge to set up an MFA application,
                 or a challenge to enter an MFA code from a registered MFA
application.
        """
        try:
            kwargs = {
                "UserPoolId": self.user_pool_id,
                "ClientId": self.client_id,
                "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",
```

```
                "AuthParameters": {"USERNAME": user_name, "PASSWORD": password},
            }
            if self.client_secret is not None:
                kwargs["AuthParameters"]["SECRET_HASH"] =
self._secret_hash(user_name)
            response = self.cognito_idp_client.admin_initiate_auth(**kwargs)
            challenge_name = response.get("ChallengeName", None)
            if challenge_name == "MFA_SETUP":
                if (
                    "SOFTWARE_TOKEN_MFA"
                    in response["ChallengeParameters"]["MFAS_CAN_SETUP"]
                ):
                    response.update(self.get_mfa_secret(response["Session"]))
                else:
                    raise RuntimeError(
                        "The user pool requires MFA setup, but the user pool is
not "
                        "configured for TOTP MFA. This example requires TOTP
MFA."
                    )
        except ClientError as err:
            logger.error(
                "Couldn't start sign in for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response
```

- For API details, see AdminInitiateAuth in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        " Set up authentication parameters
        DATA(lt_auth_params) = VALUE /aws1/
cl_cgpauthparamstype_w=>tt_authparameterstype(
          ( VALUE /aws1/cl_cgpauthparamstype_w=>ts_authparameterstype_maprow(
              key = 'USERNAME'
              value = NEW /aws1/cl_cgpauthparamstype_w( iv_user_name ) ) )
          ( VALUE /aws1/cl_cgpauthparamstype_w=>ts_authparameterstype_maprow(
              key = 'PASSWORD'
              value = NEW /aws1/cl_cgpauthparamstype_w( iv_password ) ) )
        ).

        " Add SECRET_HASH if provided
        IF iv_secret_hash IS NOT INITIAL.
          INSERT VALUE #(
            key = 'SECRET_HASH'
            value = NEW /aws1/cl_cgpauthparamstype_w( iv_secret_hash )
          ) INTO TABLE lt_auth_params.
        ENDIF.

        oo_result = lo_cgp->admininitiateauth(
          iv_userpoolid = iv_user_pool_id
          iv_clientid = iv_client_id
          iv_authflow = 'ADMIN_USER_PASSWORD_AUTH'
          it_authparameters = lt_auth_params
        ).

        DATA(lv_challenge) = oo_result->get_challengename( ).

        IF lv_challenge IS INITIAL.
          MESSAGE 'User successfully signed in.' TYPE 'I'.
        ELSE.
```

```
          MESSAGE |Authentication challenge required: { lv_challenge }.| TYPE
 'I'.
        ENDIF.

      CATCH /aws1/cx_cgpusernotfoundex INTO DATA(lo_user_ex).
        MESSAGE |User { iv_user_name } not found.| TYPE 'E'.

      CATCH /aws1/cx_cgpnotauthorizedex INTO DATA(lo_auth_ex).
        MESSAGE 'Not authorized. Check credentials.' TYPE 'E'.
    ENDTRY.
```

- For API details, see AdminInitiateAuth in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Begin an authentication session.
    ///
    /// - Parameters:
    ///   - cipClient: The `CongitoIdentityProviderClient` to use.
    ///   - clientId: The app client ID to use.
    ///   - userName: The username to check.
    ///   - password: The user's password.
    ///   - userPoolId: The user pool to use.
    ///
    /// - Returns: The session token associated with this authentication
    ///   session.
    func initiateAuth(cipClient: CognitoIdentityProviderClient, clientId: String,
                      userName: String, password: String,
                      userPoolId: String) async -> String? {
        var authParams: [String: String] = [:]
```

```
        authParams["USERNAME"] = userName
        authParams["PASSWORD"] = password

        do {
            let output = try await cipClient.adminInitiateAuth(
                input: AdminInitiateAuthInput(
                    authFlow:
CognitoIdentityProviderClientTypes.AuthFlowType.adminUserPasswordAuth,
                    authParameters: authParams,
                    clientId: clientId,
                    userPoolId: userPoolId
                )
            )

            guard let challengeName = output.challengeName else {
                print("*** Invalid response from the auth service.")
                return nil
            }

            print("=====> Response challenge is \(challengeName)")

            return output.session
        } catch _ as UserNotFoundException {
            print("*** The specified username, \(userName), doesn't exist.")
            return nil
        } catch _ as UserNotConfirmedException {
            print("*** The user \(userName) has not been confirmed.")
            return nil
        } catch {
            print("*** An unexpected error occurred.")
            return nil
        }
    }
```

- For API details, see AdminInitiateAuth in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AdminRespondToAuthChallenge` with an AWS SDK or CLI

The following code examples show how to use `AdminRespondToAuthChallenge`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/// <summary>
/// Respond to an admin authentication challenge.
/// </summary>
/// <param name="userName">The name of the user.</param>
/// <param name="clientId">The client ID.</param>
/// <param name="mfaCode">The multi-factor authentication code.</param>
/// <param name="session">The current application session.</param>
/// <param name="clientId">The user pool ID.</param>
/// <returns>The result of the authentication response.</returns>
public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
    string userName,
    string clientId,
    string mfaCode,
    string session,
    string userPoolId)
{
    Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

    var challengeResponses = new Dictionary<string, string>();
    challengeResponses.Add("USERNAME", userName);
    challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);
```

```
        var respondToAuthChallengeRequest = new
AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };

        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
        return response.AuthenticationResult;
    }
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

        Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeRequest
request;
        request.AddChallengeResponses("USERNAME", userName);
        request.AddChallengeResponses("SOFTWARE_TOKEN_MFA_CODE", mfaCode);
```

```
        request.SetChallengeName(

Aws::CognitoIdentityProvider::Model::ChallengeNameType::SOFTWARE_TOKEN_MFA);
        request.SetClientId(clientID);
        request.SetUserPoolId(userPoolID);
        request.SetSession(session);

        Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeOutcome
outcome =
                client.AdminRespondToAuthChallenge(request);

        if (outcome.IsSuccess()) {
            std::cout << "Here is the response to the challenge.\n" <<

outcome.GetResult().GetAuthenticationResult().Jsonize().View().WriteReadable()
                      << std::endl;

            accessToken =
outcome.GetResult().GetAuthenticationResult().GetAccessToken();
        }
        else {
            std::cerr << "Error with
CognitoIdentityProvider::AdminRespondToAuthChallenge. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To respond to an authentication challenge

There are many ways to respond to different authentication challenges, depending on your authentication flow, user pool configuration, and user settings. The following `admin-respond-to-auth-challenge` example provides a TOTP MFA code for diego@example.com and completes sign-in. This user pool has device remembering turned on, so the authentication result also returns a new device key.

```
aws cognito-idp admin-respond-to-auth-challenge \
    --user-pool-id us-west-2_EXAMPLE \
    --client-id 1example23456789 \
    --challenge-name SOFTWARE_TOKEN_MFA \
    --challenge-
responses USERNAME=diego@example.com,SOFTWARE_TOKEN_MFA_CODE=000000 \
    --session AYABeExample...
```

Output:

```
{
    "ChallengeParameters": {},
    "AuthenticationResult": {
        "AccessToken": "eyJra456defEXAMPLE",
        "ExpiresIn": 3600,
        "TokenType": "Bearer",
        "RefreshToken": "eyJra123abcEXAMPLE",
        "IdToken": "eyJra789ghiEXAMPLE",
        "NewDeviceMetadata": {
            "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "DeviceGroupKey": "-ExAmPlE1"
        }
    }
}
```

For more information, see Admin authentication flow in the *Amazon Cognito Developer Guide*.

- For API details, see AdminRespondToAuthChallenge in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    // Respond to an authentication challenge.
```

```
   public static void adminRespondToAuthChallenge(CognitoIdentityProviderClient
identityProviderClient,
           String userName, String clientId, String mfaCode, String session) {
       System.out.println("SOFTWARE_TOKEN_MFA challenge is generated");
       Map<String, String> challengeResponses = new HashMap<>();

       challengeResponses.put("USERNAME", userName);
       challengeResponses.put("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

       AdminRespondToAuthChallengeRequest respondToAuthChallengeRequest =
AdminRespondToAuthChallengeRequest.builder()
               .challengeName(ChallengeNameType.SOFTWARE_TOKEN_MFA)
               .clientId(clientId)
               .challengeResponses(challengeResponses)
               .session(session)
               .build();

       AdminRespondToAuthChallengeResponse respondToAuthChallengeResult =
identityProviderClient
               .adminRespondToAuthChallenge(respondToAuthChallengeRequest);

System.out.println("respondToAuthChallengeResult.getAuthenticationResult()"
               + respondToAuthChallengeResult.authenticationResult());
   }
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const adminRespondToAuthChallenge = ({
  userPoolId,
  clientId,
  username,
```

```
  totp,
  session,
}) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AdminRespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: totp,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Respond to an authentication challenge.
suspend fun adminRespondToAuthChallenge(
    userName: String,
    clientIdVal: String?,
    mfaCode: String,
    sessionVal: String?,
) {
    println("SOFTWARE_TOKEN_MFA challenge is generated")
    val challengeResponsesOb = mutableMapOf<String, String>()
```

```
    challengeResponsesOb["USERNAME"] = userName
    challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    val adminRespondToAuthChallengeRequest =
        AdminRespondToAuthChallengeRequest {
            challengeName = ChallengeNameType.SoftwareTokenMfa
            clientId = clientIdVal
            challengeResponses = challengeResponsesOb
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val respondToAuthChallengeResult =
 identityProviderClient.adminRespondToAuthChallenge(adminRespondToAuthChallengeRequest)
        println("respondToAuthChallengeResult.getAuthenticationResult()
 ${respondToAuthChallengeResult.authenticationResult}")
    }
}
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Respond to an MFA challenge by providing a code generated by an associated MFA application.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
```

```
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def respond_to_mfa_challenge(self, user_name, session, mfa_code):
        """
        Responds to a challenge for an MFA code. This completes the second step
of
        a two-factor sign-in. When sign-in is successful, it returns an access
token
        that can be used to get AWS credentials from Amazon Cognito.

        :param user_name: The name of the user who is signing in.
        :param session: Session information returned from a previous call to
initiate
                        authentication.
        :param mfa_code: A code generated by the associated MFA application.
        :return: The result of the authentication. When successful, this contains
an
                 access token for the user.
        """
        try:
            kwargs = {
                "UserPoolId": self.user_pool_id,
                "ClientId": self.client_id,
                "ChallengeName": "SOFTWARE_TOKEN_MFA",
                "Session": session,
                "ChallengeResponses": {
                    "USERNAME": user_name,
                    "SOFTWARE_TOKEN_MFA_CODE": mfa_code,
                },
            }
            if self.client_secret is not None:
                kwargs["ChallengeResponses"]["SECRET_HASH"] = self._secret_hash(
                    user_name
```

```
                )
            response =
self.cognito_idp_client.admin_respond_to_auth_challenge(**kwargs)
            auth_result = response["AuthenticationResult"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "ExpiredCodeException":
                logger.warning(
                    "Your MFA code has expired or has been used already. You
might have "
                    "to wait a few seconds until your app shows you a new code."
                )
            else:
                logger.error(
                    "Couldn't respond to mfa challenge for %s. Here's why: %s:
%s",
                    user_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return auth_result
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        " Build challenge responses
```

```
        DATA(lt_challenge_responses) = VALUE /aws1/
cl_cgpchallengerspstyp00=>tt_challengeresponsestype(
          ( VALUE /aws1/cl_cgpchallengerspstyp00=>ts_challengerspstype_maprow(
              key = 'USERNAME'
              value = NEW /aws1/cl_cgpchallengerspstyp00( iv_user_name ) ) )
          ( VALUE /aws1/cl_cgpchallengerspstyp00=>ts_challengerspstype_maprow(
              key = 'SOFTWARE_TOKEN_MFA_CODE'
              value = NEW /aws1/cl_cgpchallengerspstyp00( iv_mfa_code ) ) )
        ).

        " Add SECRET_HASH if provided
        IF iv_secret_hash IS NOT INITIAL.
          INSERT VALUE #(
            key = 'SECRET_HASH'
            value = NEW /aws1/cl_cgpchallengerspstyp00( iv_secret_hash )
          ) INTO TABLE lt_challenge_responses.
        ENDIF.

        DATA(lo_result) = lo_cgp->adminrespondtoauthchallenge(
          iv_userpoolid = iv_user_pool_id
          iv_clientid = iv_client_id
          iv_challengename = 'SOFTWARE_TOKEN_MFA'
          it_challengeresponses = lt_challenge_responses
          iv_session = iv_session
        ).

        oo_auth_result = lo_result->get_authenticationresult( ).

        IF oo_auth_result IS BOUND.
          MESSAGE 'MFA challenge completed successfully.' TYPE 'I'.
        ELSE.
          " Another challenge might be required
          DATA(lv_next_challenge) = lo_result->get_challengename( ).
          MESSAGE |Additional challenge required: { lv_next_challenge }.| TYPE
 'I'.
        ENDIF.

      CATCH /aws1/cx_cgpcodemismatchex INTO DATA(lo_code_ex).
        MESSAGE 'Invalid MFA code provided.' TYPE 'E'.

      CATCH /aws1/cx_cgpexpiredcodeex INTO DATA(lo_expired_ex).
        MESSAGE 'MFA code has expired.' TYPE 'E'.

      CATCH /aws1/cx_cgpnotauthorizedex INTO DATA(lo_auth_ex).
```

```
        MESSAGE 'Not authorized. Check MFA configuration.' TYPE 'E'.
    ENDTRY.
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Respond to the authentication challenge received from Cognito after
    /// initiating an authentication session. This involves sending a current
    /// MFA code to the service.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - userName: The user's username.
    ///   - clientId: The app client ID.
    ///   - userPoolId: The user pool to sign into.
    ///   - mfaCode: The 6-digit MFA code currently displayed by the user's
    ///     authenticator.
    ///   - session: The authentication session to continue processing.
    func adminRespondToAuthChallenge(cipClient: CognitoIdentityProviderClient,
 userName: String,
                                     clientId: String, userPoolId: String,
 mfaCode: String,
                                     session: String) async {
        print("=====> SOFTWARE_TOKEN_MFA challenge is generated...")

        var challengeResponsesOb: [String: String] = [:]
        challengeResponsesOb["USERNAME"] = userName
```

```
        challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

        do {
            let output = try await cipClient.adminRespondToAuthChallenge(
                input: AdminRespondToAuthChallengeInput(
                    challengeName:
CognitoIdentityProviderClientTypes.ChallengeNameType.softwareTokenMfa,
                    challengeResponses: challengeResponsesOb,
                    clientId: clientId,
                    session: session,
                    userPoolId: userPoolId
                )
            )

            guard let authenticationResult = output.authenticationResult else {
                print("*** Unable to get authentication result.")
                return
            }

            print("=====> Authentication result (JWTs are redacted):")
            print(authenticationResult)
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return
        } catch _ as CodeMismatchException {
            print("*** The specified MFA code doesn't match the expected value.")
            return
        } catch _ as UserNotFoundException {
            print("*** The specified username, \(userName), doesn't exist.")
            return
        } catch _ as UserNotConfirmedException {
            print("*** The user \(userName) has not been confirmed.")
            return
        } catch let error as NotAuthorizedException {
            print("*** Unauthorized access. Reason: \(error.properties.message ??
"<unknown>")")
        } catch {
            print("*** Error responding to the MFA challenge.")
            return
        }
    }
```

- For API details, see AdminRespondToAuthChallenge in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AdminSetUserPassword` with an AWS SDK or CLI

The following code examples show how to use `AdminSetUserPassword`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Write custom activity data with a Lambda function after Amazon Cognito user authentication

CLI

AWS CLI

To set a user password as an admin

The following `admin-set-user-password` example permanently sets the password for diego@example.com.

```
aws cognito-idp admin-set-user-password \
    --user-pool-id us-west-2_EXAMPLE \
    --username diego@example.com \
    --password MyExamplePassword1! \
    --permanent
```

This command produces no output.

For more information, see Passwords, password recovery, and password policies in the *Amazon Cognito Developer Guide*.

- For API details, see AdminSetUserPassword in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
```

```
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

- For API details, see AdminSetUserPassword in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `AssociateSoftwareToken` with an AWS SDK or CLI

The following code examples show how to use `AssociateSoftwareToken`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
```

```
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");

        return tokenResponse.Session;
    }
```

- For API details, see AssociateSoftwareToken in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

        Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenRequest
request;
        request.SetSession(session);
```

```
        Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenOutcome
 outcome =
                client.AssociateSoftwareToken(request);

        if (outcome.IsSuccess()) {
            std::cout
                    << "Enter this setup key into an authenticator app, for
 example Google Authenticator."
                    << std::endl;
            std::cout << "Setup key: " << outcome.GetResult().GetSecretCode()
                      << std::endl;
#ifdef USING_QR
            printAsterisksLine();
            std::cout << "\nOr scan the QR code in the file '" << QR_CODE_PATH <<
 "."
                      << std::endl;

            saveQRCode(std::string("otpauth://totp/") + userName + "?secret=" +
                       outcome.GetResult().GetSecretCode());
#endif // USING_QR
            session = outcome.GetResult().GetSession();
        }
        else {
            std::cerr << "Error with
 CognitoIdentityProvider::AssociateSoftwareToken. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see AssociateSoftwareToken in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To generate a secret key for an MFA authenticator app

The following `associate-software-token` example generates a TOTP private key for a user who has signed in and received an access token. The resulting private key can be

manually entered into an authenticator app, or applications can render it as a QR code that the user can scan.

```
aws cognito-idp associate-software-token \
    --access-token eyJra456defEXAMPLE
```

Output:

```
{
    "SecretCode": "QWERTYUIOP123456EXAMPLE"
}
```

For more information, see TOTP software token MFA in the *Amazon Cognito Developer Guide*.

- For API details, see AssociateSoftwareToken in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static String getSecretForAppMFA(CognitoIdentityProviderClient
identityProviderClient, String session) {
       AssociateSoftwareTokenRequest softwareTokenRequest =
AssociateSoftwareTokenRequest.builder()
               .session(session)
               .build();

       AssociateSoftwareTokenResponse tokenResponse = identityProviderClient
               .associateSoftwareToken(softwareTokenRequest);
       String secretCode = tokenResponse.secretCode();
       System.out.println("Enter this token into Google Authenticator");
       System.out.println(secretCode);
       return tokenResponse.session();
   }
```

- For API details, see AssociateSoftwareToken in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const associateSoftwareToken = (session) => {
  const client = new CognitoIdentityProviderClient({});
  const command = new AssociateSoftwareTokenCommand({
    Session: session,
  });

  return client.send(command);
};
```

- For API details, see AssociateSoftwareToken in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun getSecretForAppMFA(sessionVal: String?): String? {
    val softwareTokenRequest =
```

```
        AssociateSoftwareTokenRequest {
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val tokenResponse =
 identityProviderClient.associateSoftwareToken(softwareTokenRequest)
        val secretCode = tokenResponse.secretCode
        println("Enter this token into Google Authenticator")
        println(secretCode)
        return tokenResponse.session
    }
}
```

- For API details, see AssociateSoftwareToken in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
```

```
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def get_mfa_secret(self, session):
        """
        Gets a token that can be used to associate an MFA application with the
user.

        :param session: Session information returned from a previous call to
initiate
                        authentication.
        :return: An MFA token that can be used to set up an MFA application.
        """
        try:
            response =
self.cognito_idp_client.associate_software_token(Session=session)
        except ClientError as err:
            logger.error(
                "Couldn't get MFA secret. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response
```

- For API details, see AssociateSoftwareToken in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    TRY.
        DATA(lo_result) = lo_cgp->associatesoftwaretoken(
          iv_session = iv_session
        ).

        ov_secret_code = lo_result->get_secretcode( ).

        MESSAGE 'MFA secret code generated successfully.' TYPE 'I'.

      CATCH /aws1/cx_cgpresourcenotfoundex INTO DATA(lo_ex).
        MESSAGE 'Session not found or expired.' TYPE 'E'.

      CATCH /aws1/cx_cgpnotauthorizedex INTO DATA(lo_auth_ex).
        MESSAGE 'Not authorized to associate software token.' TYPE 'E'.
    ENDTRY.
```

- For API details, see AssociateSoftwareToken in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Request and display an MFA secret token that the user should enter
    /// into their authenticator to set it up for the user account.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - authSession: The authentication session to request an MFA secret
    ///     for.
    ///
    /// - Returns: A string containing the MFA secret token that should be
```

```
   ///   entered into the authenticator software.
   func getSecretForAppMFA(cipClient: CognitoIdentityProviderClient,
authSession: String?) async -> String? {
       do {
           let output = try await cipClient.associateSoftwareToken(
               input: AssociateSoftwareTokenInput(
                   session: authSession
               )
           )

           guard let secretCode = output.secretCode else {
               print("*** Unable to get the secret code")
               return nil
           }

           print("=====> Enter this token into Google Authenticator:
\(secretCode)")
           return output.session
       } catch _ as SoftwareTokenMFANotFoundException {
           print("*** The specified user pool isn't configured for MFA.")
           return nil
       } catch {
           print("*** An unexpected error occurred getting the secret for the
app's MFA.")
           return nil
       }
   }
```

- For API details, see AssociateSoftwareToken in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ConfirmDevice` with an AWS SDK or CLI

The following code examples show how to use `ConfirmDevice`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</
param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string
 deviceKey, string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }
```

- For API details, see ConfirmDevice in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To confirm a user device

The following `confirm-device` example adds a new remembered device for the current user.

```
aws cognito-idp confirm-device \
   --access-token eyJra456defEXAMPLE \
   --device-key us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111 \
   --device-secret-verifier-
config PasswordVerifier=TXlWZXJpZmllclN0cmluZw,Salt=TXlTUlBTYWx0
```

Output:

```
{
     "UserConfirmationNecessary": false
}
```

For more information, see Working with user devices in your user pool in the *Amazon Cognito Developer Guide*.

- For API details, see ConfirmDevice in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const confirmDevice = ({ deviceKey, accessToken, passwordVerifier, salt }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmDeviceCommand({
    DeviceKey: deviceKey,
    AccessToken: accessToken,
    DeviceSecretVerifierConfig: {
      PasswordVerifier: passwordVerifier,
      Salt: salt,
    },
  });
```

```
  return client.send(command);
};
```

- For API details, see ConfirmDevice in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def confirm_mfa_device(
        self,
        user_name,
        device_key,
        device_group_key,
        device_password,
```

```
        access_token,
        aws_srp,
    ):
        """
        Confirms an MFA device to be tracked by Amazon Cognito. When a device is
        tracked, its key and password can be used to sign in without requiring a
new
        MFA code from the MFA application.

        :param user_name: The user that is associated with the device.
        :param device_key: The key of the device, returned by Amazon Cognito.
        :param device_group_key: The group key of the device, returned by Amazon
Cognito.
        :param device_password: The password that is associated with the device.
        :param access_token: The user's access token.
        :param aws_srp: A class that helps with Secure Remote Password (SRP)
                        calculations. The scenario associated with this example
uses
                        the warrant package.
        :return: True when the user must confirm the device. Otherwise, False.
When
                 False, the device is automatically confirmed and tracked.
        """
        srp_helper = aws_srp.AWSSRP(
            username=user_name,
            password=device_password,
            pool_id="_",
            client_id=self.client_id,
            client_secret=None,
            client=self.cognito_idp_client,
        )
        device_and_pw = f"{device_group_key}{device_key}:{device_password}"
        device_and_pw_hash = aws_srp.hash_sha256(device_and_pw.encode("utf-8"))
        salt = aws_srp.pad_hex(aws_srp.get_random(16))
        x_value = aws_srp.hex_to_long(aws_srp.hex_hash(salt +
device_and_pw_hash))
        verifier = aws_srp.pad_hex(pow(srp_helper.val_g, x_value,
srp_helper.big_n))
        device_secret_verifier_config = {
            "PasswordVerifier": base64.standard_b64encode(
                bytearray.fromhex(verifier)
            ).decode("utf-8"),
            "Salt":
base64.standard_b64encode(bytearray.fromhex(salt)).decode("utf-8"),
```

```
        }
        try:
            response = self.cognito_idp_client.confirm_device(
                AccessToken=access_token,
                DeviceKey=device_key,
                DeviceSecretVerifierConfig=device_secret_verifier_config,
            )
            user_confirm = response["UserConfirmationNecessary"]
        except ClientError as err:
            logger.error(
                "Couldn't confirm mfa device %s. Here's why: %s: %s",
                device_key,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return user_confirm
```

- For API details, see ConfirmDevice in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ConfirmForgotPassword` with an AWS SDK or CLI

The following code examples show how to use `ConfirmForgotPassword`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Automatically migrate known users with a Lambda function

CLI

AWS CLI

To confirm a forgotten password

This example confirms a forgotten password for username diego@example.com.

Command:

```
aws cognito-idp confirm-forgot-password --client-id 3n4b5urk1ft4fl3mg5e62d9ado --
username=diego@example.com --password PASSWORD --confirmation-code CONF_CODE
```

- For API details, see ConfirmForgotPassword in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
```

```
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}
```

- For API details, see ConfirmForgotPassword in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ConfirmSignUp` with an AWS SDK or CLI

The following code examples show how to use `ConfirmSignUp`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code,
string userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            Console.WriteLine($"{userName} was confirmed");
            return true;
        }
        return false;
    }
```

- For API details, see ConfirmSignUp in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

        Aws::CognitoIdentityProvider::Model::ConfirmSignUpRequest request;
        request.SetClientId(clientID);
        request.SetConfirmationCode(confirmationCode);
        request.SetUsername(userName);

        Aws::CognitoIdentityProvider::Model::ConfirmSignUpOutcome outcome =
                client.ConfirmSignUp(request);

        if (outcome.IsSuccess()) {
            std::cout << "ConfirmSignup was Successful."
                      << std::endl;
        }
        else {
            std::cerr << "Error with CognitoIdentityProvider::ConfirmSignUp. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see ConfirmSignUp in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To confirm sign-up

This example confirms sign-up for username diego@example.com.

Command:

```
aws cognito-idp confirm-sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --
username=diego@example.com --confirmation-code CONF_CODE
```

- For API details, see ConfirmSignUp in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void confirmSignUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String code,
           String userName) {
       try {
           ConfirmSignUpRequest signUpRequest = ConfirmSignUpRequest.builder()
                   .clientId(clientId)
                   .confirmationCode(code)
                   .username(userName)
                   .build();

           identityProviderClient.confirmSignUp(signUpRequest);
           System.out.println(userName + " was confirmed");

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
```

```
        }
    }
```

- For API details, see ConfirmSignUp in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};
```

- For API details, see ConfirmSignUp in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun confirmSignUp(
    clientIdVal: String?,
    codeVal: String?,
    userNameVal: String?,
) {
    val signUpRequest =
        ConfirmSignUpRequest {
            clientId = clientIdVal
            confirmationCode = codeVal
            username = userNameVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        identityProviderClient.confirmSignUp(signUpRequest)
        println("$userNameVal  was confirmed")
    }
}
```

- For API details, see ConfirmSignUp in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
```

```
        :param client_id: The ID of a client application registered with the user
pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def confirm_user_sign_up(self, user_name, confirmation_code):
        """
        Confirms a previously created user. A user must be confirmed before they
        can sign in to Amazon Cognito.

        :param user_name: The name of the user to confirm.
        :param confirmation_code: The confirmation code sent to the user's
registered
                                  email address.
        :return: True when the confirmation succeeds.
        """
        try:
            kwargs = {
                "ClientId": self.client_id,
                "Username": user_name,
                "ConfirmationCode": confirmation_code,
            }
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            self.cognito_idp_client.confirm_sign_up(**kwargs)
        except ClientError as err:
            logger.error(
                "Couldn't confirm sign up for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return True
```

- For API details, see ConfirmSignUp in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Submit a confirmation code for the specified user. This is the code as
    /// entered by the user after they've received it by email or text
    /// message.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The app client ID the user is signing up for.
    ///   - userName: The username of the user whose code is being sent.
    ///   - code: The user's confirmation code.
    ///
    /// - Returns: `true` if the code was successfully confirmed; otherwise
 `false`.
    func confirmSignUp(cipClient: CognitoIdentityProviderClient, clientId:
 String,
                       userName: String, code: String) async -> Bool {
        do {
            _ = try await cipClient.confirmSignUp(
                input: ConfirmSignUpInput(
                    clientId: clientId,
                    confirmationCode: code,
                    username: userName
                )
            )

            print("=====> \(userName) has been confirmed.")
            return true
        } catch {
            print("=====> \(userName)'s code was entered incorrectly.")
            return false
```

```
        }
    }
```

- For API details, see ConfirmSignUp in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateUserPool` with an AWS SDK or CLI

The following code examples show how to use `CreateUserPool`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Getting started with user pools

CLI

AWS CLI

To create a minimally configured user pool

This example creates a user pool named MyUserPool using default values. There are no required attributes and no application clients. MFA and advanced security is disabled.

Command:

```
aws cognito-idp create-user-pool --pool-name MyUserPool
```

Output:

```
{
  "UserPool": {
      "SchemaAttributes": [
          {
              "Name": "sub",
              "StringAttributeConstraints": {
                  "MinLength": "1",
                  "MaxLength": "2048"
```

```
            },
            "DeveloperOnlyAttribute": false,
            "Required": true,
            "AttributeDataType": "String",
            "Mutable": false
        },
        {
            "Name": "name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "given_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "family_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "middle_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
```

```
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "nickname",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "preferred_username",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "profile",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "picture",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
```

```
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "website",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "email",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "AttributeDataType": "Boolean",
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "Name": "email_verified",
            "Mutable": true
        },
        {
            "Name": "gender",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
```

```
            "Mutable": true
        },
        {
            "Name": "birthdate",
            "StringAttributeConstraints": {
                "MinLength": "10",
                "MaxLength": "10"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "zoneinfo",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "locale",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "phone_number",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
```

```
            "Mutable": true
        },
        {
            "AttributeDataType": "Boolean",
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "Name": "phone_number_verified",
            "Mutable": true
        },
        {
            "Name": "address",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "updated_at",
            "NumberAttributeConstraints": {
                "MinValue": "0"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "Number",
            "Mutable": true
        }
    ],
    "MfaConfiguration": "OFF",
    "Name": "MyUserPool",
    "LastModifiedDate": 1547833345.777,
    "AdminCreateUserConfig": {
        "UnusedAccountValidityDays": 7,
        "AllowAdminCreateUserOnly": false
    },
    "EmailConfiguration": {},
    "Policies": {
        "PasswordPolicy": {
            "RequireLowercase": true,
            "RequireSymbols": true,
            "RequireNumbers": true,
```

```
                "MinimumLength": 8,
                "RequireUppercase": true
            }
        },
        "CreationDate": 1547833345.777,
        "EstimatedNumberOfUsers": 0,
        "Id": "us-west-2_aaaaaaaaa",
        "LambdaConfig": {}
    }
}
```

To create a user pool with two required attributes

This example creates a user pool MyUserPool. The pool is configured to accept email as a username attribute. It also sets the email source address to a validated address using Amazon Simple Email Service.

Command:

```
aws cognito-idp create-user-pool --pool-name MyUserPool --username-
attributes "email" --email-configuration=SourceArn="arn:aws:ses:us-
east-1:111111111111:identity/
jane@example.com",ReplyToEmailAddress="jane@example.com"
```

Output:

```
{
  "UserPool": {
      "SchemaAttributes": [
          {
              "Name": "sub",
              "StringAttributeConstraints": {
                  "MinLength": "1",
                  "MaxLength": "2048"
              },
              "DeveloperOnlyAttribute": false,
              "Required": true,
              "AttributeDataType": "String",
              "Mutable": false
          },
          {
              "Name": "name",
              "StringAttributeConstraints": {
```

```
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "given_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "family_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "middle_name",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "nickname",
            "StringAttributeConstraints": {
```

```
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "preferred_username",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "profile",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "picture",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "website",
            "StringAttributeConstraints": {
```

```
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "email",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "AttributeDataType": "Boolean",
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "Name": "email_verified",
            "Mutable": true
        },
        {
            "Name": "gender",
            "StringAttributeConstraints": {
                "MinLength": "0",
                "MaxLength": "2048"
            },
            "DeveloperOnlyAttribute": false,
            "Required": false,
            "AttributeDataType": "String",
            "Mutable": true
        },
        {
            "Name": "birthdate",
            "StringAttributeConstraints": {
                "MinLength": "10",
                "MaxLength": "10"
            },
            "DeveloperOnlyAttribute": false,
```

```
                "Required": false,
                "AttributeDataType": "String",
                "Mutable": true
            },
            {
                "Name": "zoneinfo",
                "StringAttributeConstraints": {
                    "MinLength": "0",
                    "MaxLength": "2048"
                },
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "AttributeDataType": "String",
                "Mutable": true
            },
            {
                "Name": "locale",
                "StringAttributeConstraints": {
                    "MinLength": "0",
                    "MaxLength": "2048"
                },
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "AttributeDataType": "String",
                "Mutable": true
            },
            {
                "Name": "phone_number",
                "StringAttributeConstraints": {
                    "MinLength": "0",
                    "MaxLength": "2048"
                },
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "AttributeDataType": "String",
                "Mutable": true
            },
            {
                "AttributeDataType": "Boolean",
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "Name": "phone_number_verified",
                "Mutable": true
            },
```

```
            {
                "Name": "address",
                "StringAttributeConstraints": {
                    "MinLength": "0",
                    "MaxLength": "2048"
                },
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "AttributeDataType": "String",
                "Mutable": true
            },
            {
                "Name": "updated_at",
                "NumberAttributeConstraints": {
                    "MinValue": "0"
                },
                "DeveloperOnlyAttribute": false,
                "Required": false,
                "AttributeDataType": "Number",
                "Mutable": true
            }
        ],
        "MfaConfiguration": "OFF",
        "Name": "MyUserPool",
        "LastModifiedDate": 1547837788.189,
        "AdminCreateUserConfig": {
            "UnusedAccountValidityDays": 7,
            "AllowAdminCreateUserOnly": false
        },
        "EmailConfiguration": {
            "ReplyToEmailAddress": "jane@example.com",
            "SourceArn": "arn:aws:ses:us-east-1:111111111111:identity/
jane@example.com"
        },
        "Policies": {
            "PasswordPolicy": {
                "RequireLowercase": true,
                "RequireSymbols": true,
                "RequireNumbers": true,
                "MinimumLength": 8,
                "RequireUppercase": true
            }
        },
        "UsernameAttributes": [
```

```
            "email"
        ],
        "CreationDate": 1547837788.189,
        "EstimatedNumberOfUsers": 0,
        "Id": "us-west-2_aaaaaaaaa",
        "LambdaConfig": {}
    }
}
```

- For API details, see CreateUserPool in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolRequest;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateUserPool {
    public static void main(String[] args) {
```

```
       final String usage = """

               Usage:
                   <userPoolName>\s

               Where:
                   userPoolName - The name to give your user pool when it's
created.
               """;

       if (args.length != 1) {
           System.out.println(usage);
           System.exit(1);
       }

       String userPoolName = args[0];
       CognitoIdentityProviderClient cognitoClient =
CognitoIdentityProviderClient.builder()
               .region(Region.US_EAST_1)
               .build();

       String id = createPool(cognitoClient, userPoolName);
       System.out.println("User pool ID: " + id);
       cognitoClient.close();
   }

   public static String createPool(CognitoIdentityProviderClient cognitoClient,
String userPoolName) {
       try {
           CreateUserPoolRequest request = CreateUserPoolRequest.builder()
                   .poolName(userPoolName)
                   .build();

           CreateUserPoolResponse response =
cognitoClient.createUserPool(request);
           return response.userPool().id();

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
       return "";
   }
```

```
}
```

- For API details, see CreateUserPool in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `CreateUserPoolClient` with an AWS SDK or CLI

The following code examples show how to use `CreateUserPoolClient`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Getting started with user pools

CLI

AWS CLI

To create a user pool client

The following `create-user-pool-client` example creates a new user pool client with a client secret, explicit read and write attributes, sign in with username-password and SRP flows, sign-in with three IdPs, access to a subset of OAuth scopes, PinPoint analytics, and an extended authentication session validity.

```
aws cognito-idp create-user-pool-client \
    --user-pool-id us-west-2_EXAMPLE \
    --client-name MyTestClient \
    --generate-secret \
    --refresh-token-validity 10 \
    --access-token-validity 60 \
    --id-token-validity 60 \
    --token-validity-units AccessToken=minutes,IdToken=minutes,RefreshToken=days
 \
    --read-attributes email phone_number email_verified phone_number_verified \
    --write-attributes email phone_number \
```

```
    --explicit-auth-
flows ALLOW_USER_PASSWORD_AUTH ALLOW_USER_SRP_AUTH ALLOW_REFRESH_TOKEN_AUTH \
    --supported-identity-providers Google Facebook MyOIDC \
    --callback-urls https://www.amazon.com https://example.com http://
localhost:8001 myapp://example \
    --allowed-o-auth-flows code implicit \
    --allowed-o-auth-scopes openid profile aws.cognito.signin.user.admin solar-
system-data/asteroids.add \
    --allowed-o-auth-flows-user-pool-client \
    --analytics-configuration ApplicationArn=arn:aws:mobiletargeting:us-
west-2:767671399759:apps/thisisanexamplepinpointapplicationid,UserDataShared=TRUE
 \
    --prevent-user-existence-errors ENABLED \
    --enable-token-revocation \
    --enable-propagate-additional-user-context-data \
    --auth-session-validity 4
```

Output:

```
{
    "UserPoolClient": {
        "UserPoolId": "us-west-2_EXAMPLE",
        "ClientName": "MyTestClient",
        "ClientId": "123abc456defEXAMPLE",
        "ClientSecret": "this1234is5678my91011example1213client1415secret",
        "LastModifiedDate": 1726788459.464,
        "CreationDate": 1726788459.464,
        "RefreshTokenValidity": 10,
        "AccessTokenValidity": 60,
        "IdTokenValidity": 60,
        "TokenValidityUnits": {
            "AccessToken": "minutes",
            "IdToken": "minutes",
            "RefreshToken": "days"
        },
        "ReadAttributes": [
            "email_verified",
            "phone_number_verified",
            "phone_number",
            "email"
        ],
        "WriteAttributes": [
            "phone_number",
```

```
            "email"
        ],
        "ExplicitAuthFlows": [
            "ALLOW_USER_PASSWORD_AUTH",
            "ALLOW_USER_SRP_AUTH",
            "ALLOW_REFRESH_TOKEN_AUTH"
        ],
        "SupportedIdentityProviders": [
            "Google",
            "MyOIDC",
            "Facebook"
        ],
        "CallbackURLs": [
            "https://example.com",
            "https://www.amazon.com",
            "myapp://example",
            "http://localhost:8001"
        ],
        "AllowedOAuthFlows": [
            "implicit",
            "code"
        ],
        "AllowedOAuthScopes": [
            "aws.cognito.signin.user.admin",
            "openid",
            "profile",
            "solar-system-data/asteroids.add"
        ],
        "AllowedOAuthFlowsUserPoolClient": true,
        "AnalyticsConfiguration": {
            "ApplicationArn": "arn:aws:mobiletargeting:us-
west-2:123456789012:apps/thisisanexamplepinpointapplicationid",
            "RoleArn": "arn:aws:iam::123456789012:role/aws-service-role/cognito-
idp.amazonaws.com/AWSServiceRoleForAmazonCognitoIdp",
            "UserDataShared": true
        },
        "PreventUserExistenceErrors": "ENABLED",
        "EnableTokenRevocation": true,
        "EnablePropagateAdditionalUserContextData": true,
        "AuthSessionValidity": 4
    }
}
```

For more information, see Application-specific settings with app clients in the *Amazon Cognito Developer Guide*.

- For API details, see CreateUserPoolClient in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientReques
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CreateUserPoolClientRespon

/**
 * A user pool client app is an application that authenticates with Amazon
 * Cognito user pools.
 * When you create a user pool, you can configure app clients that allow mobile
 * or web applications
 * to call API operations to authenticate users, manage user attributes and
 * profiles,
 * and implement sign-up and sign-in flows.
 *
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class CreateUserPoolClient {
```

```
   public static void main(String[] args) {
       final String usage = """

               Usage:
                   <clientName> <userPoolId>\s

               Where:
                   clientName - The name for the user pool client to create.
                   userPoolId - The ID for the user pool.
               """;

       if (args.length != 2) {
           System.out.println(usage);
           System.exit(1);
       }

       String clientName = args[0];
       String userPoolId = args[1];
       CognitoIdentityProviderClient cognitoClient =
CognitoIdentityProviderClient.builder()
               .region(Region.US_EAST_1)
               .build();

       createPoolClient(cognitoClient, clientName, userPoolId);
       cognitoClient.close();
   }

   public static void createPoolClient(CognitoIdentityProviderClient
cognitoClient, String clientName,
           String userPoolId) {
       try {
           CreateUserPoolClientRequest request =
CreateUserPoolClientRequest.builder()
                   .clientName(clientName)
                   .userPoolId(userPoolId)
                   .build();

           CreateUserPoolClientResponse response =
cognitoClient.createUserPoolClient(request);
           System.out.println("User pool " +
response.userPoolClient().clientName() + " created. ID: "
                   + response.userPoolClient().clientId());

       } catch (CognitoIdentityProviderException e) {
```

```
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see CreateUserPoolClient in *AWS SDK for Java 2.x API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DeleteUser` with an AWS SDK or CLI

The following code examples show how to use `DeleteUser`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Automatically confirm known users with a Lambda function
- Automatically migrate known users with a Lambda function
- Write custom activity data with a Lambda function after Amazon Cognito user authentication

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);
```

```
        Aws::CognitoIdentityProvider::Model::DeleteUserRequest request;
        request.SetAccessToken(accessToken);

        Aws::CognitoIdentityProvider::Model::DeleteUserOutcome outcome =
                client.DeleteUser(request);

        if (outcome.IsSuccess()) {
            std::cout << "The user " << userName << " was deleted."
                      << std::endl;
        }
        else {
            std::cerr << "Error with CognitoIdentityProvider::DeleteUser. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
```

- For API details, see DeleteUser in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To delete a user

This example deletes a user.

Command:

```
aws cognito-idp delete-user --access-token ACCESS_TOKEN
```

- For API details, see DeleteUser in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}
```

- For API details, see DeleteUser in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- For API details, see DeleteUser in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeTermsByClient` with an AWS SDK

The following code example shows how to use `DescribeTermsByClient`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Client token and Terms discovery

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    def describe_terms_by_client(
        self, user_pool_id: str, client_id: str, terms_name: str
    ) -> dict[str, Any]:
        """
        Looks up a Terms document associated with an app client by client ID,
        user pool ID, and terms name.

        :param user_pool_id: The ID of the user pool that contains the Terms
documents.
        :param client_id: The ID of the app client associated with the Terms.
        :param terms_name: The name of the terms document ('terms-of-use' or
                           'privacy-policy').
        :return: A dictionary with the Terms document details.
        """
        try:
            response = self.cognito_idp_client.describe_terms_by_client(
                UserPoolId=user_pool_id,
                ClientId=client_id,
                TermsName=terms_name,
            )
            terms = response.get("Terms", dict())
            logger.info(
                "Described terms '%s' for client %s in user pool %s.",
```

```
                terms_name,
                client_id,
                user_pool_id,
            )
            return terms
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "No terms documents found for client %s, user pool %s, "
                    "and terms name '%s'.",
                    client_id,
                    user_pool_id,
                    terms_name,
                )
            raise
```

- For API details, see DescribeTermsByClient in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `DescribeUserPoolClient` with an AWS SDK or CLI

The following code examples show how to use `DescribeUserPoolClient`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Client token and Terms discovery
- Getting started with user pools

CLI

AWS CLI

To describe a user pool client

This example describes a user pool client.

Command:

```
aws cognito-idp describe-user-pool-client --user-pool-id us-west-2_aaaaaaaaa --
client-id 38fjsnc484p94kpqsnet7mpld0
```

Output:

```
{
  "UserPoolClient": {
      "UserPoolId": "us-west-2_aaaaaaaaa",
      "ClientName": "MyApp",
      "ClientId": "38fjsnc484p94kpqsnet7mpld0",
      "ClientSecret": "CLIENT_SECRET",
      "LastModifiedDate": 1548108676.163,
      "CreationDate": 1548108676.163,
      "RefreshTokenValidity": 30,
      "ReadAttributes": [
          "address",
          "birthdate",
          "custom:CustomAttr1",
          "custom:CustomAttr2",
          "email",
          "email_verified",
          "family_name",
          "gender",
          "given_name",
          "locale",
          "middle_name",
          "name",
          "nickname",
          "phone_number",
          "phone_number_verified",
          "picture",
          "preferred_username",
          "profile",
          "updated_at",
          "website",
          "zoneinfo"
      ],
      "WriteAttributes": [
          "address",
          "birthdate",
          "custom:CustomAttr1",
```

```
            "custom:CustomAttr2",
            "email",
            "family_name",
            "gender",
            "given_name",
            "locale",
            "middle_name",
            "name",
            "nickname",
            "phone_number",
            "picture",
            "preferred_username",
            "profile",
            "updated_at",
            "website",
            "zoneinfo"
        ],
        "ExplicitAuthFlows": [
            "ADMIN_NO_SRP_AUTH",
            "USER_PASSWORD_AUTH"
        ],
        "AllowedOAuthFlowsUserPoolClient": false
    }
}
```

- For API details, see DescribeUserPoolClient in *AWS CLI Command Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    def describe_user_pool_client(
        self, user_pool_id: str, client_id: str
    ) -> dict[str, Any]:
        """
        Describes an app client in a user pool, including the client secret.
```

```
        :param user_pool_id: The ID of the user pool that contains the app
client.
        :param client_id: The ID of the app client to describe.
        :return: A dictionary with the app client configuration.
        """
        try:
            response = self.cognito_idp_client.describe_user_pool_client(
                UserPoolId=user_pool_id, ClientId=client_id
            )
            client_info = response.get("UserPoolClient", dict())
            logger.info(
                "Described app client %s in user pool %s.",
                client_id,
                user_pool_id,
            )
            return client_info
        except ClientError as err:
            if err.response["Error"]["Code"] == "ResourceNotFoundException":
                logger.error(
                    "The user pool %s or client %s does not exist. "
                    "Verify the stack deployed successfully and the outputs are
correct.",
                    user_pool_id,
                    client_id,
                )
            raise
```

- For API details, see DescribeUserPoolClient in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ForgotPassword` with an AWS SDK or CLI

The following code examples show how to use `ForgotPassword`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Automatically migrate known users with a Lambda function

CLI

AWS CLI

To force a password change

The following `forgot-password` example sends a message to jane@example.com to change their password.

```
aws cognito-idp forgot-password --client-id 38fjsnc484p94kpqsnet7mpld0 --
username jane@example.com
```

Output:

```
{
    "CodeDeliveryDetails": {
        "Destination": "j***@e***.com",
        "DeliveryMedium": "EMAIL",
        "AttributeName": "email"
    }
}
```

- For API details, see ForgotPassword in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
```

```
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}
```

- For API details, see ForgotPassword in *AWS SDK for Go API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `GetClientToken` with an AWS SDK

The following code example shows how to use `GetClientToken`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Client token and Terms discovery

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
def get_client_token(
    self,
    client_id: str,
    client_secret: str,
    scopes: Optional[list[str]] = None,
) -> dict[str, Any]:
    """
    Obtains an M2M access token for an app client using its client ID
    and secret, without end-user sign-in.

    :param client_id: The ID of the app client.
    :param client_secret: The client secret for the app client.
    :param scopes: Optional list of scopes in the format
                   'resource-server-identifier/scope-name'. If omitted,
                   all scopes configured for the app client are authorized.
    :return: A dictionary containing the ClientAuthenticationResult with
             AccessToken, ExpiresIn, and TokenType.
    """
    try:
        params = dict()
        params["ClientId"] = client_id
        params["Secret"] = client_secret
        if scopes is not None:
            params["Scopes"] = scopes

        response = self.cognito_idp_client.get_client_token(**params)
```

```
            auth_result = response.get("ClientAuthenticationResult", dict())
            logger.info(
                "Obtained M2M access token for client %s. Token type: %s, expires
in: %s seconds.",
                client_id,
                auth_result.get("TokenType", "Unknown"),
                auth_result.get("ExpiresIn", 0),
            )
            return auth_result
        except ClientError as err:
            if err.response["Error"]["Code"] == "NotAuthorizedException":
                logger.error(
                    "Not authorized for client %s. Verify that the client ID and
"
                    "secret are correct, and that the ALLOW_CLIENT_TOKEN_AUTH
flow "
                    "is enabled for the app client.",
                    client_id,
                )
            raise
```

- For API details, see GetClientToken in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `InitiateAuth` with an AWS SDK or CLI

The following code examples show how to use `InitiateAuth`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Automatically confirm known users with a Lambda function
- Automatically migrate known users with a Lambda function
- Getting started with user pools
- Sign up a user with a user pool that requires MFA
- Write custom activity data with a Lambda function after Amazon Cognito user authentication

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</
param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }
```

- For API details, see InitiateAuth in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To sign in a user

The following `initiate-auth` example signs in a user with the basic username-password flow and no additional challenges.

```
aws cognito-idp initiate-auth \
    --auth-flow USER_PASSWORD_AUTH \
    --client-id 1example23456789 \
    --analytics-metadata AnalyticsEndpointId=d70b2ba36a8c4dc5a04a0451aEXAMPLE \
    --auth-parameters USERNAME=testuser,PASSWORD=[Password] --user-context-
data EncodedData=mycontextdata --client-metadata MyTestKey=MyTestValue
```

Output:

```
{
    "AuthenticationResult": {
        "AccessToken": "eyJra456defEXAMPLE",
        "ExpiresIn": 3600,
        "TokenType": "Bearer",
        "RefreshToken": "eyJra123abcEXAMPLE",
        "IdToken": "eyJra789ghiEXAMPLE",
        "NewDeviceMetadata": {
            "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "DeviceGroupKey": "-v7w9UcY6"
        }
    }
}
```

For more information, see Authentication in the *Amazon Cognito Developer Guide*.

- For API details, see InitiateAuth in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
```

```
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}
```

- For API details, see InitiateAuth in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const initiateAuth = ({ username, password, clientId }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new InitiateAuthCommand({
    AuthFlow: AuthFlowType.USER_PASSWORD_AUTH,
    AuthParameters: {
      USERNAME: username,
      PASSWORD: password,
    },
    ClientId: clientId,
  });

  return client.send(command);
};
```

- For API details, see InitiateAuth in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

This example shows you how to start authentication with a tracked device. To complete sign-in, the client must respond correctly to Secure Remote Password (SRP) challenges.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def sign_in_with_tracked_device(
        self,
        user_name,
        password,
        device_key,
        device_group_key,
        device_password,
        aws_srp,
    ):
        """
```

```
        Signs in to Amazon Cognito as a user who has a tracked device. Signing in
        with a tracked device lets a user sign in without entering a new MFA
 code.

        Signing in with a tracked device requires that the client respond to the
 SRP
        protocol. The scenario associated with this example uses the warrant
 package
        to help with SRP calculations.

        For more information on SRP, see https://en.wikipedia.org/wiki/
Secure_Remote_Password_protocol.

        :param user_name: The user that is associated with the device.
        :param password: The user's password.
        :param device_key: The key of a tracked device.
        :param device_group_key: The group key of a tracked device.
        :param device_password: The password that is associated with the device.
        :param aws_srp: A class that helps with SRP calculations. The scenario
                        associated with this example uses the warrant package.
        :return: The result of the authentication. When successful, this contains
 an
                 access token for the user.
        """
        try:
            srp_helper = aws_srp.AWSSRP(
                username=user_name,
                password=device_password,
                pool_id="_",
                client_id=self.client_id,
                client_secret=None,
                client=self.cognito_idp_client,
            )

            response_init = self.cognito_idp_client.initiate_auth(
                ClientId=self.client_id,
                AuthFlow="USER_PASSWORD_AUTH",
                AuthParameters={
                    "USERNAME": user_name,
                    "PASSWORD": password,
                    "DEVICE_KEY": device_key,
                },
            )
            if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
```

```
                raise RuntimeError(
                    f"Expected DEVICE_SRP_AUTH challenge but got
{response_init['ChallengeName']}."
                )

            auth_params = srp_helper.get_auth_params()
            auth_params["DEVICE_KEY"] = device_key
            response_auth = self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_SRP_AUTH",
                ChallengeResponses=auth_params,
            )
            if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
                raise RuntimeError(
                    f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
                    f"{response_init['ChallengeName']}."
                )

            challenge_params = response_auth["ChallengeParameters"]
            challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
            cr = srp_helper.process_challenge(challenge_params, {"USERNAME":
user_name})
            cr["USERNAME"] = user_name
            cr["DEVICE_KEY"] = device_key
            response_verifier =
self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_PASSWORD_VERIFIER",
                ChallengeResponses=cr,
            )
            auth_tokens = response_verifier["AuthenticationResult"]
        except ClientError as err:
            logger.error(
                "Couldn't start client sign in for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return auth_tokens
```

- For API details, see InitiateAuth in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListUserPools` with an AWS SDK or CLI

The following code examples show how to use `ListUserPools`.

.NET

SDK for .NET (v4)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// List the Amazon Cognito user pools for an account.
    /// </summary>
    /// <returns>A list of UserPoolDescriptionType objects.</returns>
    public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
    {
        var userPools = new List<UserPoolDescriptionType>();

        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }
```

- For API details, see ListUserPools in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

To list user pools

The following `list-user-pools` example lists 3 of the available user pools in the AWS account of the current CLI credentials.

```
aws cognito-idp list-user-pools \
    --max-results 3
```

Output:

```
{
    "NextToken": "[Pagination token]",
    "UserPools": [
        {
            "CreationDate": 1681502497.741,
            "Id": "us-west-2_EXAMPLE1",
            "LambdaConfig": {
                "CustomMessage": "arn:aws:lambda:us-
east-1:123456789012:function:MyFunction",
                "PreSignUp": "arn:aws:lambda:us-
east-1:123456789012:function:MyFunction",
                "PreTokenGeneration": "arn:aws:lambda:us-
east-1:123456789012:function:MyFunction",
                "PreTokenGenerationConfig": {
                    "LambdaArn": "arn:aws:lambda:us-
east-1:123456789012:function:MyFunction",
                    "LambdaVersion": "V1_0"
                }
            },
            "LastModifiedDate": 1681502497.741,
            "Name": "user pool 1"
        },
        {
            "CreationDate": 1686064178.717,
            "Id": "us-west-2_EXAMPLE2",
            "LambdaConfig": {
```

```
            },
            "LastModifiedDate": 1686064178.873,
            "Name": "user pool 2"
        },
        {
            "CreationDate": 1627681712.237,
            "Id": "us-west-2_EXAMPLE3",
            "LambdaConfig": {
                "UserMigration": "arn:aws:lambda:us-
east-1:123456789012:function:MyFunction"
            },
            "LastModifiedDate": 1678486942.479,
            "Name": "user pool 3"
        }
    ]
}
```

For more information, see Amazon Cognito user pools in the *Amazon Cognito Developer Guide*.

- For API details, see ListUserPools in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
package main

import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Notification
 Service
// (Amazon SNS) client and list the topics in your account.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 cognitoClient := cognitoidentityprovider.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the user pools for your account.")
 var pools []types.UserPoolDescriptionType
 paginator := cognitoidentityprovider.NewListUserPoolsPaginator(
  cognitoClient, &cognitoidentityprovider.ListUserPoolsInput{MaxResults:
 aws.Int32(10)})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get user pools. Here's why: %v\n", err)
  } else {
   pools = append(pools, output.UserPools...)
  }
 }
 if len(pools) == 0 {
  fmt.Println("You don't have any user pools!")
 } else {
  for _, pool := range pools {
   fmt.Printf("\t%v: %v\n", *pool.Name, *pool.Id)
  }
 }
}
```

- For API details, see ListUserPools in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUserPoolsRequest;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListUserPools {
    public static void main(String[] args) {
        CognitoIdentityProviderClient cognitoClient =
 CognitoIdentityProviderClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listAllUserPools(cognitoClient);
        cognitoClient.close();
    }

    public static void listAllUserPools(CognitoIdentityProviderClient
 cognitoClient) {
```

```
        try {
            ListUserPoolsRequest request = ListUserPoolsRequest.builder()
                    .maxResults(10)
                    .build();

            ListUserPoolsResponse response =
 cognitoClient.listUserPools(request);
            response.userPools().forEach(userpool -> {
                System.out.println("User pool " + userpool.name() + ", User ID "
 + userpool.id());
            });

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListUserPools in *AWS SDK for Java 2.x API Reference*.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_pools(client: &Client) -> Result<(), Error> {
    let response = client.list_user_pools().max_results(10).send().await?;
    let pools = response.user_pools();
    println!("User pools:");
    for pool in pools {
        println!("  ID:              {}", pool.id().unwrap_or_default());
        println!("  Name:            {}", pool.name().unwrap_or_default());
        println!("  Lambda Config:   {:?}", pool.lambda_config().unwrap());
        println!(
            "  Last modified:   {}",
```

```
            pool.last_modified_date().unwrap().to_chrono_utc()?
        );
        println!(
            "  Creation date:   {:?}",
            pool.creation_date().unwrap().to_chrono_utc()
        );
        println!();
    }
    println!("Next token: {}", response.next_token().unwrap_or_default());

    Ok(())
}
```

- For API details, see ListUserPools in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ListUsers` with an AWS SDK or CLI

The following code examples show how to use `ListUsers`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Getting started with user pools
- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
        {
            users.AddRange(response.Users);
        }

        return users;
    }
```

- For API details, see ListUsers in *AWS SDK for .NET API Reference*.

CLI

AWS CLI

Example 1: To list users with a server-side filter

The following `list-users` example lists 3 users in the requested user pool whose email addresses begin with `testuser`.

```
aws cognito-idp list-users \
    --user-pool-id us-west-2_EXAMPLE \
    --filter email^=\"testuser\" \
    --max-items 3
```

Output:

```
{
    "PaginationToken": "efgh5678EXAMPLE",
    "Users": [
        {
            "Attributes": [
                {
                    "Name": "sub",
                    "Value": "eaad0219-2117-439f-8d46-4db20e59268f"
                },
                {
                    "Name": "email",
                    "Value": "testuser@example.com"
                }
            ],
            "Enabled": true,
            "UserCreateDate": 1682955829.578,
            "UserLastModifiedDate": 1689030181.63,
            "UserStatus": "CONFIRMED",
            "Username": "testuser"
        },
        {
            "Attributes": [
                {
                    "Name": "sub",
                    "Value": "3b994cfd-0b07-4581-be46-3c82f9a70c90"
                },
                {
                    "Name": "email",
                    "Value": "testuser2@example.com"
                }
            ],
            "Enabled": true,
            "UserCreateDate": 1684427979.201,
            "UserLastModifiedDate": 1684427979.201,
            "UserStatus": "UNCONFIRMED",
            "Username": "testuser2"
        },
        {
            "Attributes": [
                {
                    "Name": "sub",
                    "Value": "5929e0d1-4c34-42d1-9b79-a5ecacfe66f7"
                },
```

```
                {
                    "Name": "email",
                    "Value": "testuser3@example.com"
                }
            ],
            "Enabled": true,
            "UserCreateDate": 1684427823.641,
            "UserLastModifiedDate": 1684427823.641,
            "UserStatus": "UNCONFIRMED",
            "Username": "testuser3@example.com"
        }
    ]
}
```

For more information, see Managing and searching for users in the *Amazon Cognito Developer Guide*.

Example 2: To list users with a client-side filter

The following `list-users` example lists the attributes of three users who have an attribute, in this case their email address, that contains the email domain "@example.com". If other attributes contained this string, they would also be displayed. The second user has no attributes that match the query and is excluded from the displayed output, but not from the server response.

```
aws cognito-idp list-users \
    --user-pool-id us-west-2_EXAMPLE \
    --max-items 3
    --query Users\[\*\].Attributes\[\?Value\.contains\(\@\,\'@example.com\'\)\]
```

Output:

```
[
    [
        {
            "Name": "email",
            "Value": "admin@example.com"
        }
    ],
    [],
    [
        {
```

```
            "Name": "email",
            "Value": "operator@example.com"
        }
    ]
]
```

For more information, see Managing and searching for users in the *Amazon Cognito Developer Guide*.

- For API details, see ListUsers in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUsersRequest;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ListUsersResponse;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation topic:
 *
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 */
public class ListUsers {
    public static void main(String[] args) {
```

```
        final String usage = """

                Usage:
                    <userPoolId>\s

                Where:
                    userPoolId - The ID given to your user pool when it's
created.
                """;

        if (args.length != 1) {
            System.out.println(usage);
            System.exit(1);
        }

        String userPoolId = args[0];
        CognitoIdentityProviderClient cognitoClient =
CognitoIdentityProviderClient.builder()
                .region(Region.US_EAST_1)
                .build();

        listAllUsers(cognitoClient, userPoolId);
        listUsersFilter(cognitoClient, userPoolId);
        cognitoClient.close();
    }

    public static void listAllUsers(CognitoIdentityProviderClient cognitoClient,
String userPoolId) {
        try {
            ListUsersRequest usersRequest = ListUsersRequest.builder()
                    .userPoolId(userPoolId)
                    .build();

            ListUsersResponse response = cognitoClient.listUsers(usersRequest);
            response.users().forEach(user -> {
                System.out.println("User " + user.username() + " Status " +
user.userStatus() + " Created "
                        + user.userCreateDate());
            });

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
```

```
        }
    }

    // Shows how to list users by using a filter.
    public static void listUsersFilter(CognitoIdentityProviderClient
 cognitoClient, String userPoolId) {

        try {
            String filter = "email = \"tblue@noserver.com\"";
            ListUsersRequest usersRequest = ListUsersRequest.builder()
                    .userPoolId(userPoolId)
                    .filter(filter)
                    .build();

            ListUsersResponse response = cognitoClient.listUsers(usersRequest);
            response.users().forEach(user -> {
                System.out.println("User with filter applied " + user.username()
 + " Status " + user.userStatus()
                        + " Created " + user.userCreateDate());
            });

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see ListUsers in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const listUsers = ({ userPoolId }) => {
```

```
  const client = new CognitoIdentityProviderClient({});

  const command = new ListUsersCommand({
    UserPoolId: userPoolId,
  });

  return client.send(command);
};
```

- For API details, see ListUsers in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun listAllUsers(userPoolId: String) {
    val request =
        ListUsersRequest {
            this.userPoolId = userPoolId
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { cognitoClient ->
        val response = cognitoClient.listUsers(request)
        response.users?.forEach { user ->
            println("The user name is ${user.username}")
        }
    }
}
```

- For API details, see ListUsers in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def list_users(self):
        """
        Returns a list of the users in the current user pool.

        :return: The list of users.
        """
        try:
            response =
 self.cognito_idp_client.list_users(UserPoolId=self.user_pool_id)
            users = response["Users"]
        except ClientError as err:
            logger.error(
                "Couldn't list users for %s. Here's why: %s: %s",
```

```
                self.user_pool_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return users
```

- For API details, see ListUsers in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    DATA(lo_result) = lo_cgp->listusers(
      iv_userpoolid = iv_user_pool_id
    ).

    ot_users = lo_result->get_users( ).

    MESSAGE |Found { lines( ot_users ) } users in the pool.| TYPE 'I'.

  CATCH /aws1/cx_cgpresourcenotfoundex INTO DATA(lo_ex).
    MESSAGE |User pool { iv_user_pool_id } not found.| TYPE 'E'.

  CATCH /aws1/cx_cgpnotauthorizedex INTO DATA(lo_auth_ex).
    MESSAGE 'Not authorized to list users.' TYPE 'E'.
ENDTRY.
```

- For API details, see ListUsers in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        do {
            let output = try await cognitoClient.listUsers(
                input: ListUsersInput(
                    userPoolId: poolId
                )
            )

            guard let users = output.users else {
                print("No users found.")
                return
            }

            print("\(users.count) user(s) found.")
            for user in users {
                print("  \(user.username ?? "<unknown>")")
            }
        } catch _ as NotAuthorizedException {
            print("*** Please authenticate with AWS before using this command.")
            return
        } catch _ as ResourceNotFoundException {
            print("*** The specified User Pool was not found.")
            return
        } catch {
            print("*** An unexpected type of error occurred.")
            return
        }
```

- For API details, see ListUsers in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `ResendConfirmationCode` with an AWS SDK or CLI

The following code examples show how to use `ResendConfirmationCode`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);
```

```
        Console.WriteLine($"Method of delivery is
{response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }
```

- For API details, see ResendConfirmationCode in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

        Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeRequest
request;
        request.SetUsername(userName);
        request.SetClientId(clientID);

        Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeOutcome
outcome =
                client.ResendConfirmationCode(request);

        if (outcome.IsSuccess()) {
            std::cout
                    << "CognitoIdentityProvider::ResendConfirmationCode was
successful."
                    << std::endl;
```

```
        }
        else {
            std::cerr << "Error with
 CognitoIdentityProvider::ResendConfirmationCode. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see ResendConfirmationCode in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To resend a confirmation code

The following `resend-confirmation-code` example sends a confirmation code to the user `jane`.

```
aws cognito-idp resend-confirmation-code \
    --client-id 12a3b456c7de890f11g123hijk \
    --username jane
```

Output:

```
{
    "CodeDeliveryDetails": {
        "Destination": "j***@e***.com",
        "DeliveryMedium": "EMAIL",
        "AttributeName": "email"
    }
}
```

For more information, see Signing up and confirming user accounts in the *Amazon Cognito Developer Guide*.

- For API details, see ResendConfirmationCode in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void resendConfirmationCode(CognitoIdentityProviderClient
identityProviderClient, String clientId,
           String userName) {
       try {
           ResendConfirmationCodeRequest codeRequest =
ResendConfirmationCodeRequest.builder()
                   .clientId(clientId)
                   .username(userName)
                   .build();

           ResendConfirmationCodeResponse response =
identityProviderClient.resendConfirmationCode(codeRequest);
           System.out.println("Method of delivery is " +
response.codeDeliveryDetails().deliveryMediumAsString());

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see ResendConfirmationCode in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const resendConfirmationCode = ({ clientId, username }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ResendConfirmationCodeCommand({
    ClientId: clientId,
    Username: username,
  });

  return client.send(command);
};
```

- For API details, see ResendConfirmationCode in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun resendConfirmationCode(
    clientIdVal: String?,
    userNameVal: String?,
) {
    val codeRequest =
        ResendConfirmationCodeRequest {
```

```
            clientId = clientIdVal
            username = userNameVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.resendConfirmationCode(codeRequest)
        println("Method of delivery is " +
 (response.codeDeliveryDetails?.deliveryMedium))
    }
}
```

- For API details, see ResendConfirmationCode in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret
```

```
    def resend_confirmation(self, user_name):
        """
        Prompts Amazon Cognito to resend an email with a new confirmation code.

        :param user_name: The name of the user who will receive the email.
        :return: Delivery information about where the email is sent.
        """
        try:
            kwargs = {"ClientId": self.client_id, "Username": user_name}
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            response = self.cognito_idp_client.resend_confirmation_code(**kwargs)
            delivery = response["CodeDeliveryDetails"]
        except ClientError as err:
            logger.error(
                "Couldn't resend confirmation to %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return delivery
```

- For API details, see ResendConfirmationCode in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider
```

```
    /// Requests a new confirmation code be sent to the given user's contact
    /// method.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The application client ID.
    ///   - userName: The user to resend a code for.
    ///
    /// - Returns: `true` if a new code was sent successfully, otherwise
    ///   `false`.
    func resendConfirmationCode(cipClient: CognitoIdentityProviderClient,
clientId: String,
                                userName: String) async -> Bool {
        do {
            let output = try await cipClient.resendConfirmationCode(
                input: ResendConfirmationCodeInput(
                    clientId: clientId,
                    username: userName
                )
            )

            guard let deliveryMedium = output.codeDeliveryDetails?.deliveryMedium
else {
                print("*** Unable to get the delivery method for the resent
code.")
                return false
            }

            print("=====> A new code has been sent by \(deliveryMedium)")
            return true
        } catch {
            print("*** Unable to resend the confirmation code to user
\(userName).")
            return false
        }
    }
```

- For API details, see ResendConfirmationCode in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `RespondToAuthChallenge` with an AWS SDK or CLI

The following code examples show how to use `RespondToAuthChallenge`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

CLI

AWS CLI

Example 1: To respond to a NEW_PASSWORD_REQUIRED challenge

The following `respond-to-auth-challenge` example responds to a NEW_PASSWORD_REQUIRED challenge that initiate-auth returned. It sets a password for the user `jane@example.com`.

```
aws cognito-idp respond-to-auth-challenge \
    --client-id 1example23456789 \
    --challenge-name NEW_PASSWORD_REQUIRED \
    --challenge-responses USERNAME=jane@example.com,NEW_PASSWORD=[Password] \
    --session AYABeEv5HklEXAMPLE
```

Output:

```
{
    "ChallengeParameters": {},
    "AuthenticationResult": {
        "AccessToken": "ACCESS_TOKEN",
        "ExpiresIn": 3600,
        "TokenType": "Bearer",
        "RefreshToken": "REFRESH_TOKEN",
        "IdToken": "ID_TOKEN",
        "NewDeviceMetadata": {
            "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "DeviceGroupKey": "-wt2ha1Zd"
```

```
        }
    }
}
```

For more information, see Authentication in the *Amazon Cognito Developer Guide*.

Example 2: To respond to a SELECT_MFA_TYPE challenge

The following `respond-to-auth-challenge` example chooses TOTP MFA as the MFA option for the current user. The user was prompted to select an MFA type and will next be prompted to enter their MFA code.

```
aws cognito-idp respond-to-auth-challenge \
    --client-id 1example23456789
    --session AYABeEv5HklEXAMPLE
    --challenge-name SELECT_MFA_TYPE
    --challenge-responses USERNAME=testuser,ANSWER=SOFTWARE_TOKEN_MFA
```

Output:

```
{
    "ChallengeName": "SOFTWARE_TOKEN_MFA",
    "Session": "AYABeEv5HklEXAMPLE",
    "ChallengeParameters": {
        "FRIENDLY_DEVICE_NAME": "transparent"
    }
}
```

For more information, see Adding MFA in the *Amazon Cognito Developer Guide*.

Example 3: To respond to a SOFTWARE_TOKEN_MFA challenge

The following `respond-to-auth-challenge` example provides a TOTP MFA code and completes sign-in.

```
aws cognito-idp respond-to-auth-challenge \
    --client-id 1example23456789 \
    --session AYABeEv5HklEXAMPLE \
    --challenge-name SOFTWARE_TOKEN_MFA \
    --challenge-responses USERNAME=testuser,SOFTWARE_TOKEN_MFA_CODE=123456
```

Output:

```
{
    "AuthenticationResult": {
        "AccessToken": "eyJra456defEXAMPLE",
        "ExpiresIn": 3600,
        "TokenType": "Bearer",
        "RefreshToken": "eyJra123abcEXAMPLE",
        "IdToken": "eyJra789ghiEXAMPLE",
        "NewDeviceMetadata": {
            "DeviceKey": "us-west-2_a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
            "DeviceGroupKey": "-v7w9UcY6"
        }
    }
}
```

For more information, see Adding MFA in the *Amazon Cognito Developer Guide*.

- For API details, see RespondToAuthChallenge in *AWS CLI Command Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
```

```
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};
```

- For API details, see RespondToAuthChallenge in *AWS SDK for JavaScript API Reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Sign in with a tracked device. To complete sign-in, the client must respond correctly to Secure Remote Password (SRP) challenges.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
```

```
        self.client_secret = client_secret


    def sign_in_with_tracked_device(
        self,
        user_name,
        password,
        device_key,
        device_group_key,
        device_password,
        aws_srp,
    ):
        """
        Signs in to Amazon Cognito as a user who has a tracked device. Signing in
        with a tracked device lets a user sign in without entering a new MFA
 code.

        Signing in with a tracked device requires that the client respond to the
 SRP
        protocol. The scenario associated with this example uses the warrant
 package
        to help with SRP calculations.

        For more information on SRP, see https://en.wikipedia.org/wiki/
Secure_Remote_Password_protocol.

        :param user_name: The user that is associated with the device.
        :param password: The user's password.
        :param device_key: The key of a tracked device.
        :param device_group_key: The group key of a tracked device.
        :param device_password: The password that is associated with the device.
        :param aws_srp: A class that helps with SRP calculations. The scenario
                        associated with this example uses the warrant package.
        :return: The result of the authentication. When successful, this contains
 an
                 access token for the user.
        """
        try:
            srp_helper = aws_srp.AWSSRP(
                username=user_name,
                password=device_password,
                pool_id="_",
                client_id=self.client_id,
                client_secret=None,
```

```
                client=self.cognito_idp_client,
            )

            response_init = self.cognito_idp_client.initiate_auth(
                ClientId=self.client_id,
                AuthFlow="USER_PASSWORD_AUTH",
                AuthParameters={
                    "USERNAME": user_name,
                    "PASSWORD": password,
                    "DEVICE_KEY": device_key,
                },
            )
            if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
                raise RuntimeError(
                    f"Expected DEVICE_SRP_AUTH challenge but got
{response_init['ChallengeName']}."
                )

            auth_params = srp_helper.get_auth_params()
            auth_params["DEVICE_KEY"] = device_key
            response_auth = self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_SRP_AUTH",
                ChallengeResponses=auth_params,
            )
            if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
                raise RuntimeError(
                    f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
                    f"{response_init['ChallengeName']}."
                )

            challenge_params = response_auth["ChallengeParameters"]
            challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
            cr = srp_helper.process_challenge(challenge_params, {"USERNAME":
user_name})
            cr["USERNAME"] = user_name
            cr["DEVICE_KEY"] = device_key
            response_verifier =
self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_PASSWORD_VERIFIER",
                ChallengeResponses=cr,
            )
            auth_tokens = response_verifier["AuthenticationResult"]
```

```
        except ClientError as err:
            logger.error(
                "Couldn't start client sign in for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return auth_tokens
```

- For API details, see RespondToAuthChallenge in *AWS SDK for Python (Boto3) API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `SignUp` with an AWS SDK or CLI

The following code examples show how to use `SignUp`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Automatically confirm known users with a Lambda function
- Automatically migrate known users with a Lambda function
- Getting started with user pools
- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
```

```
    }
```

- For API details, see SignUp in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
       Aws::Client::ClientConfiguration clientConfig;
       // Optional: Set to the AWS Region (overrides config file).
       // clientConfig.region = "us-east-1";

   Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

       Aws::CognitoIdentityProvider::Model::SignUpRequest request;
       request.AddUserAttributes(
               Aws::CognitoIdentityProvider::Model::AttributeType().WithName(
                       "email").WithValue(email));
       request.SetUsername(userName);
       request.SetPassword(password);
       request.SetClientId(clientID);
       Aws::CognitoIdentityProvider::Model::SignUpOutcome outcome =
               client.SignUp(request);

       if (outcome.IsSuccess()) {
           std::cout << "The signup request for " << userName << " was
successful."
                     << std::endl;
       }
       else if (outcome.GetError().GetErrorType() ==

Aws::CognitoIdentityProvider::CognitoIdentityProviderErrors::USERNAME_EXISTS) {
           std::cout
```

```
                    << "The username already exists. Please enter a different
 username."
                    << std::endl;
            userExists = true;
        }
        else {
            std::cerr << "Error with CognitoIdentityProvider::SignUpRequest. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see SignUp in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To sign up a user

This example signs up jane@example.com.

Command:

```
aws cognito-idp sign-up --client-id 3n4b5urk1ft4fl3mg5e62d9ado --
username jane@example.com --password PASSWORD --user-attributes
 Name="email",Value="jane@example.com" Name="name",Value="Jane"
```

Output:

```
{
  "UserConfirmed": false,
  "UserSub": "e04d60a6-45dc-441c-a40b-e25a787d4862"
}
```

- For API details, see SignUp in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
```

```
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}
```

- For API details, see SignUp in *AWS SDK for Go API Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   public static void signUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String userName,
           String password, String email) {
       AttributeType userAttrs = AttributeType.builder()
               .name("email")
               .value(email)
               .build();

       List<AttributeType> userAttrsList = new ArrayList<>();
       userAttrsList.add(userAttrs);
       try {
           SignUpRequest signUpRequest = SignUpRequest.builder()
                   .userAttributes(userAttrsList)
                   .username(userName)
                   .clientId(clientId)
                   .password(password)
                   .build();
```

```
            identityProviderClient.signUp(signUpRequest);
            System.out.println("User has been signed up ");

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
```

- For API details, see SignUp in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};
```

- For API details, see SignUp in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
suspend fun signUp(
    clientIdVal: String?,
    userNameVal: String?,
    passwordVal: String?,
    emailVal: String?,
) {
    val userAttrs =
        AttributeType {
            name = "email"
            value = emailVal
        }

    val userAttrsList = mutableListOf<AttributeType>()
    userAttrsList.add(userAttrs)
    val signUpRequest =
        SignUpRequest {
            userAttributes = userAttrsList
            username = userNameVal
            clientId = clientIdVal
            password = passwordVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        identityProviderClient.signUp(signUpRequest)
        println("User has been signed up")
    }
}
```

- For API details, see SignUp in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def sign_up_user(self, user_name, password, user_email):
        """
        Signs up a new user with Amazon Cognito. This action prompts Amazon
 Cognito
        to send an email to the specified email address. The email contains a
 code that
        can be used to confirm the user.

        When the user already exists, the user status is checked to determine
 whether
        the user has been confirmed.

        :param user_name: The user name that identifies the new user.
```

```
        :param password: The password for the new user.
        :param user_email: The email address for the new user.
        :return: True when the user is already confirmed with Amazon Cognito.
                 Otherwise, false.
        """
        try:
            kwargs = {
                "ClientId": self.client_id,
                "Username": user_name,
                "Password": password,
                "UserAttributes": [{"Name": "email", "Value": user_email}],
            }
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            response = self.cognito_idp_client.sign_up(**kwargs)
            confirmed = response["UserConfirmed"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "UsernameExistsException":
                response = self.cognito_idp_client.admin_get_user(
                    UserPoolId=self.user_pool_id, Username=user_name
                )
                logger.warning(
                    "User %s exists and is %s.", user_name,
response["UserStatus"]
                )
                confirmed = response["UserStatus"] == "CONFIRMED"
            else:
                logger.error(
                    "Couldn't sign up %s. Here's why: %s: %s",
                    user_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return confirmed
```

- For API details, see SignUp in *AWS SDK for Python (Boto3) API Reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Create a new user in a user pool.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The ID of the app client to create a user for.
    ///   - userName: The username for the new user.
    ///   - password: The new user's password.
    ///   - email: The new user's email address.
    ///
    /// - Returns: `true` if successful; otherwise `false`.
    func signUp(cipClient: CognitoIdentityProviderClient, clientId: String,
 userName: String, password: String, email: String) async -> Bool {
        let emailAttr = CognitoIdentityProviderClientTypes.AttributeType(
            name: "email",
            value: email
        )

        let userAttrsList = [emailAttr]

        do {
            _ = try await cipClient.signUp(
                input: SignUpInput(
                    clientId: clientId,
                    password: password,
                    userAttributes: userAttrsList,
                    username: userName
                )

            )
```

```
            print("=====> User \(userName) signed up.")
        } catch _ as AWSCognitoIdentityProvider.UsernameExistsException {
            print("*** The username \(userName) already exists. Please use a
different one.")
            return false
        } catch let error as AWSCognitoIdentityProvider.InvalidPasswordException
{
            print("*** Error: The specified password is invalid. Reason:
\(error.properties.message ?? "<none available>").")
            return false
        } catch _ as AWSCognitoIdentityProvider.ResourceNotFoundException {
            print("*** Error: The specified client ID (\(clientId)) doesn't
exist.")
            return false
        } catch {
            print("*** Unexpected error: \(error)")
            return false
        }

        return true
    }
```

- For API details, see SignUp in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `UpdateUserPool` with an AWS SDK or CLI

The following code examples show how to use `UpdateUserPool`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code examples:

- Automatically confirm known users with a Lambda function
- Automatically migrate known users with a Lambda function
- Write custom activity data with a Lambda function after Amazon Cognito user authentication

CLI

AWS CLI

To update a user pool

The following `update-user-pool` example modifies a user pool with example syntax for each of the available configuration options. To update a user pool, you must specify all previously-configured options or they will reset to a default value.

```
aws cognito-idp update-user-pool --user-pool-id us-west-2_EXAMPLE \
    --policies PasswordPolicy=
\{MinimumLength=6,RequireUppercase=true,RequireLowercase=true,RequireNumbers=true,Require
 \
    --deletion-protection ACTIVE \
    --lambda-config PreSignUp="arn:aws:lambda:us-
west-2:123456789012:function:cognito-test-presignup-
function",PreTokenGeneration="arn:aws:lambda:us-
west-2:123456789012:function:cognito-test-pretoken-function" \
    --auto-verified-attributes "phone_number" "email" \
    --verification-message-template \{\"SmsMessage\":\"Your code is
 {####}"\",\"EmailMessage\":\""Your code is {####}"\",\"EmailSubject\":\""Your
 verification code"\",\"EmailMessageByLink\":\""Click {##here##} to verify
 your email address."\",\"EmailSubjectByLink\":\""Your verification link"\",
\"DefaultEmailOption\":\"CONFIRM_WITH_LINK\"\} \
    --sms-authentication-message "Your code is {####}" \
    --user-attribute-update-settings
 AttributesRequireVerificationBeforeUpdate="email","phone_number" \
    --mfa-configuration "OPTIONAL" \
    --device-
configuration ChallengeRequiredOnNewDevice=true,DeviceOnlyRememberedOnUserPrompt=true
 \
    --email-configuration SourceArn="arn:aws:ses:us-
west-2:123456789012:identity/admin@example.com",ReplyToEmailAddress="amdin
+noreply@example.com",EmailSendingAccount=DEVELOPER,From="admin@amazon.com",Configuration
configuration-set" \
    --sms-configuration SnsCallerArn="arn:aws:iam::123456789012:role/service-
role/SNS-SMS-Role",ExternalId="12345",SnsRegion="us-west-2" \
    --admin-create-user-config
 AllowAdminCreateUserOnly=false,InviteMessageTemplate=\{SMSMessage=\""Welcome
 {username}. Your confirmation code is {####}"\",EmailMessage=\""Welcome
 {username}. Your confirmation code is {####}"\",EmailSubject=\""Welcome to
 MyMobileGame"\"\} \
```

```
    --user-pool-tags "Function"="MyMobileGame","Developers"="Berlin" \
    --admin-create-user-config
 AllowAdminCreateUserOnly=false,InviteMessageTemplate=\{SMSMessage=\""Welcome
 {username}. Your confirmation code is {####}"\",EmailMessage=\""Welcome
 {username}. Your confirmation code is {####}"\",EmailSubject=\""Welcome to
 MyMobileGame"\"\} \
    --user-pool-add-ons AdvancedSecurityMode="AUDIT" \
    --account-recovery-setting RecoveryMechanisms=
\[\{Priority=1,Name="verified_email"\},
\{Priority=2,Name="verified_phone_number"\}\]
```

This command produces no output.

For more information, see Updating user pool configuration in the *Amazon Cognito Developer Guide*.

- For API details, see UpdateUserPool in *AWS CLI Command Reference*.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}
```

```
// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
```

```
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}
```

- For API details, see UpdateUserPool in *AWS SDK for Go API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
```

```
      UserPoolId: userPoolId,
      LambdaConfig: {
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- For API details, see UpdateUserPool in *AWS SDK for JavaScript API Reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use `VerifySoftwareToken` with an AWS SDK or CLI

The following code examples show how to use `VerifySoftwareToken`.

Action examples are code excerpts from larger programs and must be run in context. You can see this action in context in the following code example:

- Sign up a user with a user pool that requires MFA

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
    /// <summary>
    /// Verify the TOTP and register for MFA.
```

```
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
    }
```

- For API details, see VerifySoftwareToken in *AWS SDK for .NET API Reference*.

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

    Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);

        Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenRequest request;
        request.SetUserCode(userCode);
```

```
        request.SetSession(session);

        Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenOutcome outcome =
                client.VerifySoftwareToken(request);

        if (outcome.IsSuccess()) {
            std::cout << "Verification of the code was successful."
                      << std::endl;
            session = outcome.GetResult().GetSession();
        }
        else {
            std::cerr << "Error with
 CognitoIdentityProvider::VerifySoftwareToken. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
```

- For API details, see VerifySoftwareToken in *AWS SDK for C++ API Reference*.

CLI

AWS CLI

To confirm registration of a TOTP authenticator

The following `verify-software-token` example completes TOTP registration for the current user.

```
aws cognito-idp verify-software-token \
    --access-token eyJra456defEXAMPLE \
    --user-code 123456
```

Output:

```
{
    "Status": "SUCCESS"
}
```

For more information, see Adding MFA to a user pool in the *Amazon Cognito Developer Guide*.

- For API details, see VerifySoftwareToken in *AWS CLI Command Reference*.

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
   // Verify the TOTP and register for MFA.
   public static void verifyTOTP(CognitoIdentityProviderClient
identityProviderClient, String session, String code) {
       try {
           VerifySoftwareTokenRequest tokenRequest =
VerifySoftwareTokenRequest.builder()
                   .userCode(code)
                   .session(session)
                   .build();

           VerifySoftwareTokenResponse verifyResponse =
identityProviderClient.verifySoftwareToken(tokenRequest);
           System.out.println("The status of the token is " +
verifyResponse.statusAsString());

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }
```

- For API details, see VerifySoftwareToken in *AWS SDK for Java 2.x API Reference*.

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});

  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- For API details, see VerifySoftwareToken in *AWS SDK for JavaScript API Reference*.

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
// Verify the TOTP and register for MFA.
suspend fun verifyTOTP(
    sessionVal: String?,
    codeVal: String?,
) {
    val tokenRequest =
        VerifySoftwareTokenRequest {
            userCode = codeVal
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val verifyResponse =
 identityProviderClient.verifySoftwareToken(tokenRequest)
        println("The status of the token is ${verifyResponse.status}")
    }
}
```

- For API details, see VerifySoftwareToken in *AWS SDK for Kotlin API reference*.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def verify_mfa(self, session, user_code):
        """
        Verify a new MFA application that is associated with a user.

        :param session: Session information returned from a previous call to
 initiate
                        authentication.
        :param user_code: A code generated by the associated MFA application.
        :return: Status that indicates whether the MFA application is verified.
        """
        try:
            response = self.cognito_idp_client.verify_software_token(
                Session=session, UserCode=user_code
            )
        except ClientError as err:
            logger.error(
                "Couldn't verify MFA. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response
```

- For API details, see VerifySoftwareToken in *AWS SDK for Python (Boto3) API Reference*.

SAP ABAP

SDK for SAP ABAP

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
TRY.
    DATA(lo_result) = lo_cgp->verifysoftwaretoken(
      iv_session = iv_session
      iv_usercode = iv_user_code
    ).

    ov_status = lo_result->get_status( ).

    IF ov_status = 'SUCCESS'.
      MESSAGE 'MFA token verified successfully.' TYPE 'I'.
    ELSE.
      MESSAGE |MFA verification status: { ov_status }.| TYPE 'I'.
    ENDIF.

  CATCH /aws1/cx_cgpcodemismatchex INTO DATA(lo_code_ex).
    MESSAGE 'Invalid MFA code provided.' TYPE 'E'.

  CATCH /aws1/cx_cgpenbsoftwaretokmf00 INTO DATA(lo_enabled_ex).
    MESSAGE 'Software token MFA is already enabled.' TYPE 'E'.
ENDTRY.
```

- For API details, see VerifySoftwareToken in *AWS SDK for SAP ABAP API reference*.

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import AWSClientRuntime
import AWSCognitoIdentityProvider

    /// Confirm that the user's TOTP authenticator is configured correctly by
    /// sending a code to it to check that it matches successfully.
    ///
    /// - Parameters:
    ///   - cipClient: The `CongnitoIdentityProviderClient` to use.
    ///   - session: An authentication session previously returned by an
    ///     `associateSoftwareToken()` call.
    ///   - mfaCode: The 6-digit code currently displayed by the user's
    ///     authenticator, as provided by the user.
    func verifyTOTP(cipClient: CognitoIdentityProviderClient, session: String?,
 mfaCode: String?) async {
        do {
            let output = try await cipClient.verifySoftwareToken(
                input: VerifySoftwareTokenInput(
                    session: session,
                    userCode: mfaCode
                )
            )

            guard let tokenStatus = output.status else {
                print("*** Unable to get the token's status.")
                return
            }
            print("=====> The token's status is: \(tokenStatus)")
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return
        } catch _ as CodeMismatchException {
            print("*** The specified MFA code doesn't match the expected value.")
```

```
            return
        } catch _ as UserNotFoundException {
            print("*** The specified username doesn't exist.")
            return
        } catch _ as UserNotConfirmedException {
            print("*** The user has not been confirmed.")
            return
        } catch {
            print("*** Error verifying the MFA token!")
            return
        }
    }
```

- For API details, see VerifySoftwareToken in *AWS SDK for Swift API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Scenarios for Amazon Cognito Identity Provider using AWS SDKs

The following code examples show you how to implement common scenarios in Amazon Cognito Identity Provider with AWS SDKs. These scenarios show you how to accomplish specific tasks by calling multiple functions within Amazon Cognito Identity Provider or combined with other AWS services. Each scenario includes a link to the complete source code, where you can find instructions on how to set up and run the code.

Scenarios target an intermediate level of experience to help you understand service actions in context.

Examples

- Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK
- Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK
- Get M2M client tokens and discover Terms with Amazon Cognito using an AWS SDK
- Getting started with user pools
- Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK
- Use Amazon Cognito identity pools and authentication flows

- Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK

Automatically confirm known Amazon Cognito users with a Lambda function using an AWS SDK

The following code examples show how to automatically confirm known Amazon Cognito users with a Lambda function.

- Configure a user pool to call a Lambda function for the `PreSignUp` trigger.
- Sign up a user with Amazon Cognito.
- The Lambda function scans a DynamoDB table and automatically confirms known users.
- Sign in as the new user, then clean up resources.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)
```

```
// AutoConfirm separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type AutoConfirm struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewAutoConfirm constructs a new auto confirm runner.
func NewAutoConfirm(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) AutoConfirm {
 scenario := AutoConfirm{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddPreSignUpTrigger adds a Lambda handler as an invocation target for the
 PreSignUp trigger.
func (runner *AutoConfirm) AddPreSignUpTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the PreSignUp trigger from
 Cognito.\n" +
  "This trigger happens when a user signs up, and lets your function take action
 before the main Cognito\n" +
  "sign up processing occurs.\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PreSignUp, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the PreSignUp
 trigger.\n",
  functionArn, userPoolId)
```

```
}

// SignUpUser signs up a user from the known user table with a password you
 specify.
func (runner *AutoConfirm) SignUpUser(ctx context.Context, clientId string,
 usersTable string) (string, string) {
 log.Println("Let's sign up a user to your Cognito user pool. When the user's
 email matches an email in the\n" +
  "DynamoDB known users table, it is automatically verified and the user is
 confirmed.")

 knownUsers, err := runner.helper.GetKnownUsers(ctx, usersTable)
 if err != nil {
  panic(err)
 }
 userChoice := runner.questioner.AskChoice("Which user do you want to use?\n",
 knownUsers.UserNameList())
 user := knownUsers.Users[userChoice]

 var signedUp bool
 var userConfirmed bool
 password := runner.questioner.AskPassword("Enter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !signedUp {
  log.Printf("Signing up user '%v' with email '%v' to Cognito.\n", user.UserName,
 user.UserEmail)
  userConfirmed, err = runner.cognitoActor.SignUp(ctx, clientId, user.UserName,
 password, user.UserEmail)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("Enter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   signedUp = true
  }
 }
 log.Printf("User %v signed up, confirmed = %v.\n", user.UserName, userConfirmed)

 log.Println(strings.Repeat("-", 88))
```

```
 return user.UserName, password
}

// SignInUser signs in a user.
func (runner *AutoConfirm) SignInUser(ctx context.Context, clientId string,
 userName string, password string) string {
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 log.Printf("Let's sign in as %v...\n", userName)
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 log.Println(strings.Repeat("-", 88))
 return *authResult.AccessToken
}

// Run runs the scenario.
func (runner *AutoConfirm) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])

 runner.AddPreSignUpTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["AutoConfirmFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers, actions.PreSignUp)
 userName, password := runner.SignUpUser(ctx, stackOutputs["UserPoolClientId"],
 stackOutputs["TableName"])
```

```
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["AutoConfirmFunction"])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
  runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password))

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Handle the `PreSignUp` trigger with a Lambda function.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
```

```
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PreSignUp event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be confirmed and verified.
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsPreSignup) (events.CognitoEventUserPoolsPreSignup,
 error) {
 log.Printf("Received presignup from %v for user '%v'", event.TriggerSource,
 event.UserName)
 if event.TriggerSource != "PreSignUp_SignUp" {
  // Other trigger sources, such as PreSignUp_AdminInitiateAuth, ignore the
 response from this handler.
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserEmail: event.Request.UserAttributes["email"],
 }
 log.Printf("Looking up email %v in table %v.\n", user.UserEmail, tableName)
 output, err := h.dynamoClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key:       user.GetKey(),
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Error looking up email %v.\n", user.UserEmail)
  return event, err
 }
 if output.Item == nil {
  log.Printf("Email %v not found. Email verification is required.\n",
 user.UserEmail)
  return event, err
 }

 err = attributevalue.UnmarshalMap(output.Item, &user)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB item. Here's why: %v\n", err)
```

```
  return event, err
 }

 if user.UserName != event.UserName {
  log.Printf("UserEmail %v found, but stored UserName '%v' does not match
 supplied UserName '%v'. Verification is required.\n",
   user.UserEmail, user.UserName, event.UserName)
 } else {
  log.Printf("UserEmail %v found with matching UserName %v. User is confirmed.
\n", user.UserEmail, user.UserName)
  event.Response.AutoConfirmUser = true
  event.Response.AutoVerifyEmail = true
 }

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
```

```
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
```

```
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)
```

```
type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
```

```
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}
```

```
// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}
```

```
// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
```

```
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
```

```
 }
 return err
}
```

Create a struct that wraps DynamoDB actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
```

```
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
```

```
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Create a struct that wraps CloudFormation actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
```

```
 }
 return stackOutputs
}
```

Clean up resources.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
```

```
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Configure an interactive "Scenario" run. The JavaScript (v3) examples share a Scenario runner to streamline complex examples. The complete source code is on GitHub.

```
import { AutoConfirm } from "./scenario-auto-confirm.js";

/**
 * The context is passed to every scenario. Scenario steps
 * will modify the context.
 */
const context = {
  errors: [],
  users: [
    {
      UserName: "test_user_1",
      UserEmail: "test_email_1@example.com",
    },
    {
      UserName: "test_user_2",
      UserEmail: "test_email_2@example.com",
    },
    {
      UserName: "test_user_3",
      UserEmail: "test_email_3@example.com",
    },
  ],
};

/**
 * Three Scenarios are created for the workflow. A Scenario is an orchestration
 class
 * that simplifies running a series of steps.
 */
```

```
export const scenarios = {
  // Demonstrate automatically confirming known users in a database.
  "auto-confirm": AutoConfirm(context),
};

// Call function if run directly
import { fileURLToPath } from "node:url";
import { parseScenarioArgs } from "@aws-doc-sdk-examples/lib/scenario/index.js";

if (process.argv[1] === fileURLToPath(import.meta.url)) {
  parseScenarioArgs(scenarios, {
    name: "Cognito user pools and triggers",
    description:
      "Demonstrate how to use the AWS SDKs to customize Amazon Cognito
 authentication behavior.",
  });
}
```

This Scenario demonstrates auto-confirming a known user. It orchestrates the example steps.

```
import { wait } from "@aws-doc-sdk-examples/lib/utils/util-timers.js";
import {
  Scenario,
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";

import {
  getStackOutputs,
  logCleanUpReminder,
  promptForStackName,
  promptForStackRegion,
  skipWhenErrors,
} from "./steps-common.js";
import { populateTable } from "./actions/dynamodb-actions.js";
import {
  addPreSignUpHandler,
  deleteUser,
  getUser,
  signIn,
```

```
  signUpUser,
} from "./actions/cognito-actions.js";
import {
  getLatestLogStreamForLambda,
  getLogEvents,
} from "./actions/cloudwatch-logs-actions.js";

/**
 * @typedef {{
 *   errors: Error[],
 *   password: string,
 *   users: { UserName: string, UserEmail: string }[],
 *   selectedUser?: string,
 *   stackName?: string,
 *   stackRegion?: string,
 *   token?: string,
 *   confirmDeleteSignedInUser?: boolean,
 *   TableName?: string,
 *   UserPoolClientId?: string,
 *   UserPoolId?: string,
 *   UserPoolArn?: string,
 *   AutoConfirmHandlerArn?: string,
 *   AutoConfirmHandlerName?: string
 * }} State
 */

const greeting = new ScenarioOutput(
  "greeting",
  (/** @type {State} */ state) => `This demo will populate some users into the \
database created as part of the "${state.stackName}" stack. \
Then the AutoConfirmHandler will be linked to the PreSignUp \
trigger from Cognito. Finally, you will choose a user to sign up.`,
  { skipWhen: skipWhenErrors },
);

const logPopulatingUsers = new ScenarioOutput(
  "logPopulatingUsers",
  "Populating the DynamoDB table with some users.",
  { skipWhenErrors: skipWhenErrors },
);

const logPopulatingUsersComplete = new ScenarioOutput(
  "logPopulatingUsersComplete",
  "Done populating users.",
```

```
  { skipWhen: skipWhenErrors },
);

const populateUsers = new ScenarioAction(
  "populateUsers",
  async (/** @type {State} */ state) => {
    const [_, err] = await populateTable({
      region: state.stackRegion,
      tableName: state.TableName,
      items: state.users,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTrigger = new ScenarioOutput(
  "logSetupSignUpTrigger",
  "Setting up the PreSignUp trigger for the Cognito User Pool.",
  { skipWhen: skipWhenErrors },
);

const setupSignUpTrigger = new ScenarioAction(
  "setupSignUpTrigger",
  async (/** @type {State} */ state) => {
    const [_, err] = await addPreSignUpHandler({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      handlerArn: state.AutoConfirmHandlerArn,
    });
    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const logSetupSignUpTriggerComplete = new ScenarioOutput(
```

```
  "logSetupSignUpTriggerComplete",
  (
    /** @type {State} */ state,
  ) => `The lambda function "${state.AutoConfirmHandlerName}" \
has been configured as the PreSignUp trigger handler for the user pool
 "${state.UserPoolId}".`,
  { skipWhen: skipWhenErrors },
);

const selectUser = new ScenarioInput(
  "selectedUser",
  "Select a user to sign up.",
  {
    type: "select",
    choices: (/** @type {State} */ state) => state.users.map((u) => u.UserName),
    skipWhen: skipWhenErrors,
    default: (/** @type {State} */ state) => state.users[0].UserName,
  },
);

const checkIfUserAlreadyExists = new ScenarioAction(
  "checkIfUserAlreadyExists",
  async (/** @type {State} */ state) => {
    const [user, err] = await getUser({
      region: state.stackRegion,
      userPoolId: state.UserPoolId,
      username: state.selectedUser,
    });

    if (err?.name === "UserNotFoundException") {
      // Do nothing. We're not expecting the user to exist before
      // sign up is complete.
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    if (user) {
      state.errors.push(
        new Error(
```

```
          `The user "${state.selectedUser}" already exists in the user pool
 "${state.UserPoolId}".`,
        ),
      );
    }
  },
  {
    skipWhen: skipWhenErrors,
  },
);

const createPassword = new ScenarioInput(
  "password",
  "Enter a password that has at least eight characters, uppercase, lowercase,
 numbers and symbols.",
  { type: "password", skipWhen: skipWhenErrors, default: "Abcd1234!" },
);

const logSignUpExistingUser = new ScenarioOutput(
  "logSignUpExistingUser",
  (/** @type {State} */ state) => `Signing up user "${state.selectedUser}".`,
  { skipWhen: skipWhenErrors },
);

const signUpExistingUser = new ScenarioAction(
  "signUpExistingUser",
  async (/** @type {State} */ state) => {
    const signUp = (password) =>
      signUpUser({
        region: state.stackRegion,
        userPoolClientId: state.UserPoolClientId,
        username: state.selectedUser,
        email: state.users.find((u) => u.UserName === state.selectedUser)
          .UserEmail,
        password,
      });

    let [_, err] = await signUp(state.password);

    while (err?.name === "InvalidPasswordException") {
      console.warn("The password you entered was invalid.");
      await createPassword.handle(state);
      [_, err] = await signUp(state.password);
    }
```

```
    if (err) {
      state.errors.push(err);
    }
  },
  { skipWhen: skipWhenErrors },
);

const logSignUpExistingUserComplete = new ScenarioOutput(
  "logSignUpExistingUserComplete",
  (/** @type {State} */ state) =>
    `"${state.selectedUser} was signed up successfully.`,
  { skipWhen: skipWhenErrors },
);

const logLambdaLogs = new ScenarioAction(
  "logLambdaLogs",
  async (/** @type {State} */ state) => {
    console.log(
      "Waiting a few seconds to let Lambda write to CloudWatch Logs...\n",
    );
    await wait(10);

    const [logStream, logStreamErr] = await getLatestLogStreamForLambda({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
    });
    if (logStreamErr) {
      state.errors.push(logStreamErr);
      return;
    }

    console.log(
      `Getting some recent events from log stream "${logStream.logStreamName}"`,
    );
    const [logEvents, logEventsErr] = await getLogEvents({
      functionName: state.AutoConfirmHandlerName,
      region: state.stackRegion,
      eventCount: 10,
      logStreamName: logStream.logStreamName,
    });
    if (logEventsErr) {
      state.errors.push(logEventsErr);
      return;
```

```
    }

    console.log(logEvents.map((ev) => `\t${ev.message}`).join(""));
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUser = new ScenarioOutput(
  "logSignInUser",
  (/** @type {State} */ state) => `Let's sign in as ${state.selectedUser}`,
  { skipWhen: skipWhenErrors },
);

const signInUser = new ScenarioAction(
  "signInUser",
  async (/** @type {State} */ state) => {
    const [response, err] = await signIn({
      region: state.stackRegion,
      clientId: state.UserPoolClientId,
      username: state.selectedUser,
      password: state.password,
    });

    if (err?.name === "PasswordResetRequiredException") {
      state.errors.push(new Error("Please reset your password."));
      return;
    }

    if (err) {
      state.errors.push(err);
      return;
    }

    state.token = response?.AuthenticationResult?.AccessToken;
  },
  { skipWhen: skipWhenErrors },
);

const logSignInUserComplete = new ScenarioOutput(
  "logSignInUserComplete",
  (/** @type {State} */ state) =>
    `Successfully signed in. Your access token starts with:
 ${state.token.slice(0, 11)}`,
  { skipWhen: skipWhenErrors },
```

```
);

const confirmDeleteSignedInUser = new ScenarioInput(
  "confirmDeleteSignedInUser",
  "Do you want to delete the currently signed in user?",
  { type: "confirm", skipWhen: skipWhenErrors },
);

const deleteSignedInUser = new ScenarioAction(
  "deleteSignedInUser",
  async (/** @type {State} */ state) => {
    const [_, err] = await deleteUser({
      region: state.stackRegion,
      accessToken: state.token,
    });

    if (err) {
      state.errors.push(err);
    }
  },
  {
    skipWhen: (/** @type {State} */ state) =>
      skipWhenErrors(state) || !state.confirmDeleteSignedInUser,
  },
);

const logErrors = new ScenarioOutput(
  "logErrors",
  (/** @type {State}*/ state) => {
    const errorList = state.errors
      .map((err) => ` - ${err.name}: ${err.message}`)
      .join("\n");
    return `Scenario errors found:\n${errorList}`;
  },
  {
    // Don't log errors when there aren't any!
    skipWhen: (/** @type {State} */ state) => state.errors.length === 0,
  },
);

export const AutoConfirm = (context) =>
  new Scenario(
    "AutoConfirm",
    [
```

```
      promptForStackName,
      promptForStackRegion,
      getStackOutputs,
      greeting,
      logPopulatingUsers,
      populateUsers,
      logPopulatingUsersComplete,
      logSetupSignUpTrigger,
      setupSignUpTrigger,
      logSetupSignUpTriggerComplete,
      selectUser,
      checkIfUserAlreadyExists,
      createPassword,
      logSignUpExistingUser,
      signUpExistingUser,
      logSignUpExistingUserComplete,
      logLambdaLogs,
      logSignInUser,
      signInUser,
      logSignInUserComplete,
      confirmDeleteSignedInUser,
      deleteSignedInUser,
      logCleanUpReminder,
      logErrors,
    ],
    context,
  );
```

These are steps that are shared with other Scenarios.

```
import {
  ScenarioAction,
  ScenarioInput,
  ScenarioOutput,
} from "@aws-doc-sdk-examples/lib/scenario/scenario.js";
import { getCfnOutputs } from "@aws-doc-sdk-examples/lib/sdk/cfn-outputs.js";

export const skipWhenErrors = (state) => state.errors.length > 0;

export const getStackOutputs = new ScenarioAction(
  "getStackOutputs",
  async (state) => {
```

```
    if (!state.stackName || !state.stackRegion) {
      state.errors.push(
        new Error(
          "No stack name or region provided. The stack name and \
region are required to fetch CFN outputs relevant to this example.",
        ),
      );
      return;
    }

    const outputs = await getCfnOutputs(state.stackName, state.stackRegion);
    Object.assign(state, outputs);
  },
);

export const promptForStackName = new ScenarioInput(
  "stackName",
  "Enter the name of the stack you deployed earlier.",
  { type: "input", default: "PoolsAndTriggersStack" },
);

export const promptForStackRegion = new ScenarioInput(
  "stackRegion",
  "Enter the region of the stack you deployed earlier.",
  { type: "input", default: "us-east-1" },
);

export const logCleanUpReminder = new ScenarioOutput(
  "logCleanUpReminder",
  "All done. Remember to run 'cdk destroy' to teardown the stack.",
  { skipWhen: skipWhenErrors },
);
```

A handler for the `PreSignUp` trigger with a Lambda function.

```
import type { PreSignUpTriggerEvent, Handler } from "aws-lambda";
import type { UserRepository } from "./user-repository";
import { DynamoDBUserRepository } from "./user-repository";

export class PreSignUpHandler {
  private userRepository: UserRepository;
```

```
  constructor(userRepository: UserRepository) {
    this.userRepository = userRepository;
  }

  private isPreSignUpTriggerSource(event: PreSignUpTriggerEvent): boolean {
    return event.triggerSource === "PreSignUp_SignUp";
  }

  private getEventUserEmail(event: PreSignUpTriggerEvent): string {
    return event.request.userAttributes.email;
  }

  async handlePreSignUpTriggerEvent(
    event: PreSignUpTriggerEvent,
  ): Promise<PreSignUpTriggerEvent> {
    console.log(
      `Received presignup from ${event.triggerSource} for user
'${event.userName}'`,
    );

    if (!this.isPreSignUpTriggerSource(event)) {
      return event;
    }

    const eventEmail = this.getEventUserEmail(event);
    console.log(`Looking up email ${eventEmail}.`);
    const storedUserInfo =
      await this.userRepository.getUserInfoByEmail(eventEmail);

    if (!storedUserInfo) {
      console.log(
        `Email ${eventEmail} not found. Email verification is required.`,
      );
      return event;
    }

    if (storedUserInfo.UserName !== event.userName) {
      console.log(
        `UserEmail ${eventEmail} found, but stored UserName
'${storedUserInfo.UserName}' does not match supplied UserName
'${event.userName}'. Verification is required.`,
      );
    } else {
      console.log(
```

```
        `UserEmail ${eventEmail} found with matching UserName
 ${storedUserInfo.UserName}. User is confirmed.`,
      );
      event.response.autoConfirmUser = true;
      event.response.autoVerifyEmail = true;
    }
    return event;
  }
}

const createPreSignUpHandler = (): PreSignUpHandler => {
  const tableName = process.env.TABLE_NAME;
  if (!tableName) {
    throw new Error("TABLE_NAME environment variable is not set");
  }

  const userRepository = new DynamoDBUserRepository(tableName);
  return new PreSignUpHandler(userRepository);
};

export const handler: Handler = async (event: PreSignUpTriggerEvent) => {
  const preSignUpHandler = createPreSignUpHandler();
  return preSignUpHandler.handlePreSignUpTriggerEvent(event);
};
```

Module of CloudWatch Logs actions.

```
import {
  CloudWatchLogsClient,
  GetLogEventsCommand,
  OrderBy,
  paginateDescribeLogStreams,
} from "@aws-sdk/client-cloudwatch-logs";

/**
 * Get the latest log stream for a Lambda function.
 * @param {{ functionName: string, region: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").LogStream | null,
 unknown]>}
 */
export const getLatestLogStreamForLambda = async ({ functionName, region }) => {
```

```
  try {
    const logGroupName = `/aws/lambda/${functionName}`;
    const cwlClient = new CloudWatchLogsClient({ region });
    const paginator = paginateDescribeLogStreams(
      { client: cwlClient },
      {
        descending: true,
        limit: 1,
        orderBy: OrderBy.LastEventTime,
        logGroupName,
      },
    );

    for await (const page of paginator) {
      return [page.logStreams[0], null];
    }
  } catch (err) {
    return [null, err];
  }
};

/**
 * Get the log events for a Lambda function's log stream.
 * @param {{
 *   functionName: string,
 *   logStreamName: string,
 *   eventCount: number,
 *   region: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cloudwatch-logs").OutputLogEvent[]
 | null, unknown]>}
 */
export const getLogEvents = async ({
  functionName,
  logStreamName,
  eventCount,
  region,
}) => {
  try {
    const cwlClient = new CloudWatchLogsClient({ region });
    const logGroupName = `/aws/lambda/${functionName}`;
    const response = await cwlClient.send(
      new GetLogEventsCommand({
        logStreamName: logStreamName,
```

```
        limit: eventCount,
        logGroupName: logGroupName,
      }),
    );

    return [response.events, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module of Amazon Cognito actions.

```
import {
  AdminGetUserCommand,
  CognitoIdentityProviderClient,
  DeleteUserCommand,
  InitiateAuthCommand,
  SignUpCommand,
  UpdateUserPoolCommand,
} from "@aws-sdk/client-cognito-identity-provider";

/**
 * Connect a Lambda function to the PreSignUp trigger for a Cognito user pool
 * @param {{ region: string, userPoolId: string, handlerArn: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").UpdateUserPoolCommandOutput | null, unknown]>}
 */
export const addPreSignUpHandler = async ({
  region,
  userPoolId,
  handlerArn,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const command = new UpdateUserPoolCommand({
      UserPoolId: userPoolId,
      LambdaConfig: {
```

```
        PreSignUp: handlerArn,
      },
    });

    const response = await cognitoClient.send(command);
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Attempt to register a user to a user pool with a given username and password.
 * @param {{
 *   region: string,
 *   userPoolClientId: string,
 *   username: string,
 *   email: string,
 *   password: string
 * }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").SignUpCommandOutput | null, unknown]>}
 */
export const signUpUser = async ({
  region,
  userPoolClientId,
  username,
  email,
  password,
}) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({
      region,
    });

    const response = await cognitoClient.send(
      new SignUpCommand({
        ClientId: userPoolClientId,
        Username: username,
        Password: password,
        UserAttributes: [{ Name: "email", Value: email }],
      }),
    );
    return [response, null];
```

```
  } catch (err) {
    return [null, err];
  }
};

/**
 * Sign in a user to Amazon Cognito using a username and password authentication
 flow.
 * @param {{ region: string, clientId: string, username: string, password:
 string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").InitiateAuthCommandOutput | null, unknown]>}
 */
export const signIn = async ({ region, clientId, username, password }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new InitiateAuthCommand({
        AuthFlow: "USER_PASSWORD_AUTH",
        ClientId: clientId,
        AuthParameters: { USERNAME: username, PASSWORD: password },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Retrieve an existing user from a user pool.
 * @param {{ region: string, userPoolId: string, username: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").AdminGetUserCommandOutput | null, unknown]>}
 */
export const getUser = async ({ region, userPoolId, username }) => {
  try {
    const cognitoClient = new CognitoIdentityProviderClient({ region });
    const response = await cognitoClient.send(
      new AdminGetUserCommand({
        UserPoolId: userPoolId,
        Username: username,
      }),
    );
```

```
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};

/**
 * Delete the signed-in user. Useful for allowing a user to delete their
 * own profile.
 * @param {{ region: string, accessToken: string }} config
 * @returns {Promise<[import("@aws-sdk/client-cognito-identity-
provider").DeleteUserCommandOutput | null, unknown]>}
 */
export const deleteUser = async ({ region, accessToken }) => {
  try {
    const client = new CognitoIdentityProviderClient({ region });
    const response = await client.send(
      new DeleteUserCommand({ AccessToken: accessToken }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

Module of DynamoDB actions.

```
import { DynamoDBClient } from "@aws-sdk/client-dynamodb";
import {
  BatchWriteCommand,
  DynamoDBDocumentClient,
} from "@aws-sdk/lib-dynamodb";

/**
 * Populate a DynamoDB table with provide items.
 * @param {{ region: string, tableName: string, items: Record<string,
 unknown>[] }} config
 * @returns {Promise<[import("@aws-sdk/lib-dynamodb").BatchWriteCommandOutput |
 null, unknown]>}
 */
export const populateTable = async ({ region, tableName, items }) => {
```

```
  try {
    const ddbClient = new DynamoDBClient({ region });
    const docClient = DynamoDBDocumentClient.from(ddbClient);
    const response = await docClient.send(
      new BatchWriteCommand({
        RequestItems: {
          [tableName]: items.map((item) => ({
            PutRequest: {
              Item: item,
            },
          })),
        },
      }),
    );
    return [response, null];
  } catch (err) {
    return [null, err];
  }
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - DeleteUser
  - InitiateAuth
  - SignUp
  - UpdateUserPool

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Automatically migrate known Amazon Cognito users with a Lambda function using an AWS SDK

The following code example shows how to automatically migrate known Amazon Cognito users with a Lambda function.

- Configure a user pool to call a Lambda function for the `MigrateUser` trigger.
- Sign in to Amazon Cognito with a username and email that is not in the user pool.

- The Lambda function scans a DynamoDB table and automatically migrates known users to the user pool.
- Perform the forgot password flow to reset the password for the migrated user.
- Sign in as the new user, then clean up resources.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "errors"
 "fmt"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// MigrateUser separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type MigrateUser struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}
```

```
// NewMigrateUser constructs a new migrate user runner.
func NewMigrateUser(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) MigrateUser {
 scenario := MigrateUser{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddMigrateUserTrigger adds a Lambda handler as an invocation target for the
 MigrateUser trigger.
func (runner *MigrateUser) AddMigrateUserTrigger(ctx context.Context, userPoolId
 string, functionArn string) {
 log.Printf("Let's add a Lambda function to handle the MigrateUser trigger from
 Cognito.\n" +
  "This trigger happens when an unknown user signs in, and lets your function
 take action before Cognito\n" +
  "rejects the user.\n\n")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.UserMigration, HandlerArn:
 aws.String(functionArn)})
 if err != nil {
  panic(err)
 }
 log.Printf("Lambda function %v added to user pool %v to handle the MigrateUser
 trigger.\n",
  functionArn, userPoolId)

 log.Println(strings.Repeat("-", 88))
}

// SignInUser adds a new user to the known users table and signs that user in to
 Amazon Cognito.
func (runner *MigrateUser) SignInUser(ctx context.Context, usersTable string,
 clientId string) (bool, actions.User) {
 log.Println("Let's sign in a user to your Cognito user pool. When the username
 and email matches an entry in the\n" +
```

```
 "DynamoDB known users table, the email is automatically verified and the user
is migrated to the Cognito user pool.")

user := actions.User{}
user.UserName = runner.questioner.Ask("\nEnter a username:")
user.UserEmail = runner.questioner.Ask("\nEnter an email that you own. This
email will be used to confirm user migration\n" +
 "during this example:")

runner.helper.AddKnownUser(ctx, usersTable, user)

var err error
var resetRequired *types.PasswordResetRequiredException
var authResult *types.AuthenticationResultType
signedIn := false
for !signedIn && resetRequired == nil {
 log.Printf("Signing in to Cognito as user '%v'. The expected result is a
PasswordResetRequiredException.\n\n", user.UserName)
 authResult, err = runner.cognitoActor.SignIn(ctx, clientId, user.UserName, "_")
 if err != nil {
  if errors.As(err, &resetRequired) {
   log.Printf("\nUser '%v' is not in the Cognito user pool but was found in the
DynamoDB known users table.\n"+
    "User migration is started and a password reset is required.",
user.UserName)
  } else {
   panic(err)
  }
 } else {
  log.Printf("User '%v' successfully signed in. This is unexpected and probably
means you have not\n"+
   "cleaned up a previous run of this scenario, so the user exist in the Cognito
user pool.\n"+
   "You can continue this example and select to clean up resources, or manually
remove\n"+
   "the user from your user pool and try again.", user.UserName)
  runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
*authResult.AccessToken)
  signedIn = true
 }
}

log.Println(strings.Repeat("-", 88))
return resetRequired != nil, user
```

```
}

// ResetPassword starts a password recovery flow.
func (runner *MigrateUser) ResetPassword(ctx context.Context, clientId string,
 user actions.User) {
 wantCode := runner.questioner.AskBool(fmt.Sprintf("In order to migrate the user
 to Cognito, you must be able to receive a confirmation\n"+
  "code by email at %v. Do you want to send a code (y/n)?", user.UserEmail), "y")
 if !wantCode {
  log.Println("To complete this example and successfully migrate a user to
 Cognito, you must enter an email\n" +
   "you own that can receive a confirmation code.")
  return
 }
 codeDelivery, err := runner.cognitoActor.ForgotPassword(ctx, clientId,
 user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("\nA confirmation code has been sent to %v.",
 *codeDelivery.Destination)
 code := runner.questioner.Ask("Check your email and enter it here:")

 confirmed := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
  "(the password will not display as you type):", 8)
 for !confirmed {
  log.Printf("\nConfirming password reset for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.ConfirmForgotPassword(ctx, clientId, code,
 user.UserName, password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   confirmed = true
  }
 }
 log.Printf("User '%v' successfully confirmed and migrated.\n", user.UserName)
 log.Println("Signing in with your username and password...")
```

```
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, user.UserName,
 password)
 if err != nil {
  panic(err)
 }
 log.Printf("Successfully signed in. Your access token starts with: %v...\n",
 (*authResult.AccessToken)[:10])
 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)

 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *MigrateUser) Run(ctx context.Context, stackName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]

 runner.AddMigrateUserTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["MigrateUserFunctionArn"])
 runner.resources.triggers = append(runner.resources.triggers,
 actions.UserMigration)
 resetNeeded, user := runner.SignInUser(ctx, stackOutputs["TableName"],
 stackOutputs["UserPoolClientId"])
 if resetNeeded {
  runner.helper.ListRecentLogEvents(ctx, stackOutputs["MigrateUserFunction"])
  runner.ResetPassword(ctx, stackOutputs["UserPoolClientId"], user)
 }
```

```
 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Handle the `MigrateUser` trigger with a Lambda function.

```
import (
 "context"
 "log"
 "os"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
)

const TABLE_NAME = "TABLE_NAME"

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string `dynamodbav:"UserName"`
 UserEmail string `dynamodbav:"UserEmail"`
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the MigrateUser event by looking up a user in an Amazon
 DynamoDB table and
// specifying whether they should be migrated to the user pool.
```

```
func (h *handler) HandleRequest(ctx context.Context, event
 events.CognitoEventUserPoolsMigrateUser)
 (events.CognitoEventUserPoolsMigrateUser, error) {
 log.Printf("Received migrate trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 if event.TriggerSource != "UserMigration_Authentication" {
  return event, nil
 }
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName: event.UserName,
 }
 log.Printf("Looking up user '%v' in table %v.\n", user.UserName, tableName)
 filterEx := expression.Name("UserName").Equal(expression.Value(user.UserName))
 expr, err := expression.NewBuilder().WithFilter(filterEx).Build()
 if err != nil {
  log.Printf("Error building expression to query for user '%v'.\n",
 user.UserName)
  return event, err
 }
 output, err := h.dynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName:                 aws.String(tableName),
  FilterExpression:          expr.Filter(),
  ExpressionAttributeNames:  expr.Names(),
  ExpressionAttributeValues: expr.Values(),
 })
 if err != nil {
  log.Printf("Error looking up user '%v'.\n", user.UserName)
  return event, err
 }
 if len(output.Items) == 0 {
  log.Printf("User '%v' not found, not migrating user.\n", user.UserName)
  return event, err
 }

 var users []UserInfo
 err = attributevalue.UnmarshalListOfMaps(output.Items, &users)
 if err != nil {
  log.Printf("Couldn't unmarshal DynamoDB items. Here's why: %v\n", err)
  return event, err
 }

 user = users[0]
```

```
 log.Printf("UserName '%v' found with email %v. User is migrated and must reset
 password.\n", user.UserName, user.UserEmail)
 event.CognitoEventUserPoolsMigrateUserResponse.UserAttributes =
 map[string]string{
  "email":          user.UserEmail,
  "email_verified": "true", // email_verified is required for the forgot password
 flow.
 }
 event.CognitoEventUserPoolsMigrateUserResponse.FinalUserStatus =
 "RESET_REQUIRED"
 event.CognitoEventUserPoolsMigrateUserResponse.MessageAction = "SUPPRESS"

 return event, err
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
```

```
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}
```

```
// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
```

```
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}

// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
```

```
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
```

```
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}
```

```
// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}



// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
```

```
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}



// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
```

```
}
```

Create a struct that wraps DynamoDB actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}
```

```
// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
   log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
   return err
  }
  writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
```

```
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
```

```
  events = output.Events
 }
 return events, err
}
```

Create a struct that wraps CloudFormation actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string

type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Clean up resources.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger

 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()
```

```
 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
   log.Println("Couldn't update Cognito triggers during cleanup.")
   panic(err)
  }
  log.Println("Removed Cognito triggers from user pool.")
 } else {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - ConfirmForgotPassword
  - DeleteUser
  - ForgotPassword
  - InitiateAuth
  - SignUp
  - UpdateUserPool

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Get M2M client tokens and discover Terms with Amazon Cognito using an AWS SDK

The following code example shows how to use Amazon Cognito machine-to-machine (M2M) authorization and Terms discovery.

- Deploy a CloudFormation stack with Cognito resources (user pool, resource server, app client, Terms documents).
- Retrieve the app client secret with DescribeUserPoolClient.
- Obtain M2M access tokens with GetClientToken.
- Discover Terms documents with DescribeTermsByClient.
- Clean up by deleting the CloudFormation stack.

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
class ClientTokenAndTermsScenario:
    """
    Runs an interactive scenario that demonstrates the Amazon Cognito
    GetClientToken and DescribeTermsByClient APIs.
    """

    def __init__(
        self,
        cognito_wrapper: CognitoIdentityProviderWrapper,
        cfn_client: BaseClient,
```

```
    ):
        """
        :param cognito_wrapper: An instance of the CognitoIdentityProviderWrapper
class.
        :param cfn_client: A Boto3 CloudFormation client.
        """
        self.cognito_wrapper = cognito_wrapper
        self.cfn_client = cfn_client
        self.stack_name = None
        self.user_pool_id = None
        self.client_id = None
        self.client_secret = None

    def run(self) -> None:
        """Runs the scenario from start to finish."""
        print(SEPARATOR)
        print("Welcome to the Amazon Cognito Client Token and Terms Scenario.")
        print(SEPARATOR)
        print("This example creates Cognito resources in a CloudFormation stack,
then")
        print("demonstrates the new GetClientToken and DescribeTermsByClient
APIs.")
        print()

        try:
            self.setup()
            self.get_client_secret()
            self.demonstrate_get_client_token()
            self.demonstrate_describe_terms()
        except Exception:
            logger.exception("An error occurred during the scenario.")
        finally:
            self.cleanup()

        print(SEPARATOR)
        print("Amazon Cognito Client Token and Terms scenario completed.")

    # ------------------------------------------------------------------
    # Phase 1: Setup - deploy CloudFormation stack
    # ------------------------------------------------------------------
    def setup(self) -> None:
        """Deploys the CloudFormation prerequisite stack and retrieves
outputs."""
        stack_name = q.ask(
```

```
            f"\nEnter a name for the CloudFormation stack [{DEFAULT_STACK_NAME}]:
"
        )
        if not stack_name:
            stack_name = DEFAULT_STACK_NAME

        # Check whether the stack name already exists.
        while self._stack_exists(stack_name):
            print(f"A stack named '{stack_name}' already exists.")
            stack_name = q.ask("Enter a different stack name: ")

        self.stack_name = stack_name

        template_body = self._read_template()
        print(f"Deploying CloudFormation stack: {self.stack_name}")
        self.cfn_client.create_stack(
            StackName=self.stack_name,
            TemplateBody=template_body,
            Capabilities=["CAPABILITY_IAM"],
        )
        print(f"CloudFormation stack creation started: {self.stack_name}")
        print("Waiting for CloudFormation stack creation to complete...")

        waiter = self.cfn_client.get_waiter("stack_create_complete")
        try:
            waiter.wait(
                StackName=self.stack_name,
                WaiterConfig={"Delay": 10, "MaxAttempts": 60},
            )
        except WaiterError:
            logger.error("Stack creation failed or timed out.")
            self._print_stack_events()
            raise

        print("CloudFormation stack creation complete.")
        self._retrieve_stack_outputs()
        print(SEPARATOR)

    def _stack_exists(self, stack_name: str) -> bool:
        """Returns True if a stack with the given name exists and is not
deleted."""
        if not stack_name:
            return False
        try:
```

```
            response = self.cfn_client.describe_stacks(StackName=stack_name)
            stacks = response.get("Stacks", list())
            for stack in stacks:
                status = stack.get("StackStatus", "")
                if "DELETE_COMPLETE" not in status:
                    return True
            return False
        except ClientError as err:
            if "does not exist" in str(err):
                return False
            raise

    def _read_template(self) -> str:
        """Reads the CloudFormation template from the local file system."""
        template_path = os.path.join(
            os.path.dirname(os.path.abspath(__file__)), CFN_TEMPLATE_PATH
        )
        if not os.path.exists(template_path):
            raise FileNotFoundError(
                f"CloudFormation template not found at {template_path}. "
                "Ensure the resources/cfn_template.yaml file exists."
            )
        with open(template_path, "r") as f:
            return f.read()

    def _retrieve_stack_outputs(self) -> None:
        """Retrieves and displays outputs from the CloudFormation stack."""
        response = self.cfn_client.describe_stacks(StackName=self.stack_name)
        stacks = response.get("Stacks", list())
        if not stacks:
            raise RuntimeError(f"Stack '{self.stack_name}' not found.")

        outputs = stacks[0].get("Outputs", list())
        for output in outputs:
            key = output.get("OutputKey", "")
            value = output.get("OutputValue", "")
            print(f"Stack output {key}: {value}")
            if key == "UserPoolId":
                self.user_pool_id = value
            elif key == "ClientId":
                self.client_id = value

        if not self.user_pool_id or not self.client_id:
            raise RuntimeError(
```

```
                "Failed to retrieve UserPoolId or ClientId from stack outputs."
            )

    def _print_stack_events(self) -> None:
        """Prints recent stack events to help diagnose failures."""
        try:
            response =
self.cfn_client.describe_stack_events(StackName=self.stack_name)
            events = response.get("StackEvents", list())
            for event in events[:10]:
                status = event.get("ResourceStatus", "")
                reason = event.get("ResourceStatusReason", "")
                resource = event.get("LogicalResourceId", "")
                if "FAILED" in status:
                    print(f"  FAILED: {resource} - {reason}")
        except ClientError:
            logger.warning("Could not retrieve stack events.")

    # ------------------------------------------------------------------
    # Phase 2: Retrieve the app client secret
    # ------------------------------------------------------------------
    def get_client_secret(self) -> None:
        """Calls DescribeUserPoolClient to retrieve the client secret."""
        print(SEPARATOR)
        print("Retrieving the app client secret...")
        client_info = self.cognito_wrapper.describe_user_pool_client(
            self.user_pool_id, self.client_id
        )
        self.client_secret = client_info.get("ClientSecret", "")
        if not self.client_secret:
            raise RuntimeError(
                "Client secret not found. Ensure the app client was created "
                "with GenerateSecret: true."
            )
        print("Client secret retrieved successfully.")
        print(SEPARATOR)

    # ------------------------------------------------------------------
    # Phase 3: Demonstrate GetClientToken
    # ------------------------------------------------------------------
    def demonstrate_get_client_token(self) -> None:
        """Demonstrates obtaining M2M access tokens with GetClientToken."""
        print(SEPARATOR)
        print("Obtaining an M2M access token with GetClientToken...")
```

```
        print()

        # 3a: Get token with all scopes (no scopes param = all configured scopes)
        print("Step 1: Request a token with all configured scopes.")
        auth_result = self.cognito_wrapper.get_client_token(
            client_id=self.client_id,
            client_secret=self.client_secret,
        )
        self._display_token_result(auth_result, "all scopes")

        q.ask("\nPress Enter to continue...")

        # 3b: Get token with specific scopes
        print()
        print("Step 2: Request a token with a specific scope (my-m2m-api/read).")
        scoped_result = self.cognito_wrapper.get_client_token(
            client_id=self.client_id,
            client_secret=self.client_secret,
            scopes=["my-m2m-api/read"],
        )
        self._display_token_result(scoped_result, "my-m2m-api/read")

        print()
        print("Token was successfully scoped to the requested permissions.")
        print(SEPARATOR)

    @staticmethod
    def _display_token_result(auth_result: dict, scope_description: str) -> None:
        """Displays token metadata."""
        token_type = auth_result.get("TokenType", "Unknown")
        expires_in = auth_result.get("ExpiresIn", 0)
        access_token = auth_result.get("AccessToken", "")
        preview = access_token[:20] + "..." if len(access_token) > 20 else
access_token
        print(f"  Scope:      {scope_description}")
        print(f"  Token type: {token_type}")
        print(f"  Expires in: {expires_in} seconds")
        print(f"  Token preview: {preview}")

    # ------------------------------------------------------------------
    # Phase 4: Demonstrate DescribeTermsByClient
    # ------------------------------------------------------------------
    def demonstrate_describe_terms(self) -> None:
```

```
        """Demonstrates discovering Terms documents with
DescribeTermsByClient."""
        print(SEPARATOR)
        print("Discovering Terms documents with DescribeTermsByClient...")
        print()

        # 4a: Look up terms-of-use
        print("Looking up terms-of-use for the app client...")
        terms_of_use = self.cognito_wrapper.describe_terms_by_client(
            user_pool_id=self.user_pool_id,
            client_id=self.client_id,
            terms_name="terms-of-use",
        )
        self._display_terms(terms_of_use)

        q.ask("\nPress Enter to continue...")
        print()

        # 4b: Look up privacy-policy
        print("Looking up privacy-policy for the app client...")
        privacy_policy = self.cognito_wrapper.describe_terms_by_client(
            user_pool_id=self.user_pool_id,
            client_id=self.client_id,
            terms_name="privacy-policy",
        )
        self._display_terms(privacy_policy)

        print()
        print(
            "DescribeTermsByClient lets you discover which Terms documents are "
            "associated with a client without knowing the Terms resource ID up
front."
        )
        print(SEPARATOR)

    @staticmethod
    def _display_terms(terms: dict) -> None:
        """Displays a Terms document's details."""
        print(f"  Terms ID:          {terms.get('TermsId', 'N/A')}")
        print(f"  Terms name:        {terms.get('TermsName', 'N/A')}")
        print(f"  Client ID:         {terms.get('ClientId', 'N/A')}")
        print(f"  User pool ID:      {terms.get('UserPoolId', 'N/A')}")
        print(f"  Enforcement:       {terms.get('Enforcement', 'N/A')}")
        print(f"  Terms source:      {terms.get('TermsSource', 'N/A')}")
```

```
        links = terms.get("Links", dict())
        if links:
            print("  Links:")
            for lang_tag, url in links.items():
                print(f"    {lang_tag}: {url}")
        # Cognito returns CreationDate / LastModifiedDate as datetime objects
        # (boto3 deserializes the timestamps), so display them directly.
        creation_date = terms.get("CreationDate", None)
        if creation_date is not None:
            print(f"  Created:           {creation_date}")
        modified_date = terms.get("LastModifiedDate", None)
        if modified_date is not None:
            print(f"  Last modified:     {modified_date}")

    # ------------------------------------------------------------------
    # Phase 5: Cleanup - delete CloudFormation stack
    # ------------------------------------------------------------------
    def cleanup(self) -> None:
        """Prompts the user and deletes the CloudFormation stack."""
        if self.stack_name is None:
            return

        print(SEPARATOR)
        answer = q.ask(
            "Do you want to delete the CloudFormation stack and all resources?
(y/n) "
        )
        if answer.lower() != "y":
            print(
                f"Resources will remain. You can delete the stack later "
                f"through the AWS Console or CLI."
            )
            print(f"Stack name: {self.stack_name}")
            return

        print(
            f"CloudFormation stack '{self.stack_name}' is being deleted. "
            f"This may take a few minutes."
        )
        try:
            self.cfn_client.delete_stack(StackName=self.stack_name)
            waiter = self.cfn_client.get_waiter("stack_delete_complete")
            print("Waiting for CloudFormation stack deletion to complete...")
            waiter.wait(
```

```
                StackName=self.stack_name,
                WaiterConfig={"Delay": 10, "MaxAttempts": 60},
            )
            print(f"CloudFormation stack '{self.stack_name}' has been deleted.")
            print("All Cognito resources have been removed.")
        except (ClientError, WaiterError):
            logger.warning(
                "Stack deletion encountered an error. Attempting force delete."
            )
            try:
                self.cfn_client.delete_stack(
                    StackName=self.stack_name,
                    DeletionMode="FORCE_DELETE_STACK",
                )
                print("Force delete initiated. Please verify in the console.")
            except ClientError:
                logger.error(
                    "Force delete also failed. Clean up manually via the
 console."
                )
```

Create a class that wraps Amazon Cognito Identity Provider operations.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito Identity Provider actions."""

    def __init__(self, cognito_idp_client: BaseClient):
        """
        Initializes the CognitoIdentityProviderWrapper with an Amazon Cognito
        Identity Provider client.

        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        """
        self.cognito_idp_client = cognito_idp_client
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - DescribeTermsByClient

- DescribeUserPoolClient
- GetClientToken

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Getting started with user pools

The following code example shows how to:

- Create a user pool
- Create an app client
- Set up a domain for your user pool
- Create a user as an administrator
- Enable self-registration
- List users in the user pool
- Clean up resources

Bash

AWS CLI with Bash script

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Sample developer tutorials repository.

```
#!/bin/bash

# Amazon Cognito User Pools Getting Started Script
# This script creates and configures an Amazon Cognito user pool with an app
 client

set -euo pipefail
```

```
# Security: Set restrictive umask
umask 0077

# Set up logging with secure permissions
LOG_FILE="cognito-user-pool-setup.log"
touch "$LOG_FILE"
chmod 600 "$LOG_FILE"
exec > >(tee -a "$LOG_FILE") 2>&1

echo "Starting Amazon Cognito User Pool setup script at $(date)"
echo "All commands and outputs will be logged to $LOG_FILE"

# Function to check for errors in command output
check_error() {
  local output=$1
  local cmd=$2

  if echo "$output" | grep -qi "error\|failed"; then
    echo "ERROR: Command failed: $cmd" >&2
    echo "Output: $output" >&2
    cleanup_on_error
    exit 1
  fi
}

# Function to check AWS CLI return code
check_aws_error() {
  local exit_code=$?
  local cmd=$1

  if [ $exit_code -ne 0 ]; then
    echo "ERROR: AWS CLI command failed with exit code $exit_code: $cmd" >&2
    cleanup_on_error
    exit "$exit_code"
  fi
}

# Function to clean up resources on error
cleanup_on_error() {
  echo "Error encountered. Attempting to clean up resources..." >&2

  if [ -n "${DOMAIN_NAME:-}" ] && [ -n "${USER_POOL_ID:-}" ]; then
    echo "Deleting user pool domain: $DOMAIN_NAME" >&2
    aws cognito-idp delete-user-pool-domain \
```

```
      --user-pool-id "$USER_POOL_ID" \
      --domain "$DOMAIN_NAME" 2>/dev/null || true
  fi

  if [ -n "${USER_POOL_ID:-}" ]; then
    echo "Deleting user pool: $USER_POOL_ID" >&2
    aws cognito-idp delete-user-pool \
      --user-pool-id "$USER_POOL_ID" 2>/dev/null || true
  fi
}

# Set trap for cleanup on exit
trap cleanup_on_error EXIT ERR

# Validate AWS CLI is installed and configured
if ! command -v aws &> /dev/null; then
  echo "ERROR: AWS CLI is not installed" >&2
  exit 1
fi

# Validate jq is installed
if ! command -v jq &> /dev/null; then
  echo "ERROR: jq is not installed" >&2
  exit 1
fi

# Validate openssl is installed
if ! command -v openssl &> /dev/null; then
  echo "ERROR: openssl is not installed" >&2
  exit 1
fi

if ! aws sts get-caller-identity &> /dev/null; then
  echo "ERROR: AWS CLI is not configured or credentials are invalid" >&2
  exit 1
fi

# Get the current AWS region
AWS_REGION=$(aws configure get region)
if [ -z "$AWS_REGION" ]; then
  AWS_REGION="us-east-1" # Default region if not configured
fi
echo "Using AWS Region: $AWS_REGION"
```

```
# Validate region format
if ! [[ "$AWS_REGION" =~ ^[a-z]{2}-[a-z]+-[0-9]{1}$ ]]; then
  echo "ERROR: Invalid AWS region format: $AWS_REGION" >&2
  exit 1
fi

# Generate random identifier for resource names using secure method
RANDOM_ID=$(openssl rand -hex 6)
if [ -z "$RANDOM_ID" ]; then
  echo "ERROR: Failed to generate random identifier" >&2
  exit 1
fi

USER_POOL_NAME="MyUserPool-${RANDOM_ID}"
APP_CLIENT_NAME="MyAppClient-${RANDOM_ID}"
DOMAIN_NAME="my-auth-domain-${RANDOM_ID}"

# Validate resource names don't exceed limits
if [ ${#USER_POOL_NAME} -gt 128 ]; then
  echo "ERROR: User pool name exceeds maximum length of 128 characters" >&2
  exit 1
fi

if [ ${#APP_CLIENT_NAME} -gt 128 ]; then
  echo "ERROR: App client name exceeds maximum length of 128 characters" >&2
  exit 1
fi

echo "Using random identifier: $RANDOM_ID"
echo "User pool name: $USER_POOL_NAME"
echo "App client name: $APP_CLIENT_NAME"
echo "Domain name: $DOMAIN_NAME"

# Step 1: Create a User Pool
echo "Creating user pool..."
USER_POOL_OUTPUT=$(aws cognito-idp create-user-pool \
  --pool-name "$USER_POOL_NAME" \
  --auto-verified-attributes email \
  --username-attributes email \
  --policies '{"PasswordPolicy":
{"MinimumLength":12,"RequireUppercase":true,"RequireLowercase":true,"RequireNumbers":true
 \
  --schema '[{"Name":"email","Required":true,"Mutable":true}]' \
  --mfa-configuration OFF \
```

```
  --user-attribute-update-settings '{"AttributesRequireVerificationBeforeUpdate":
["email"]}' \
  --account-recovery-setting
 'RecoveryMechanisms=[{Name=verified_email,Priority=1}]' \
  --deletion-protection INACTIVE \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "create-user-pool"

# Extract the User Pool ID using jq for safety
USER_POOL_ID=$(echo "$USER_POOL_OUTPUT" | jq -r '.UserPool.Id // empty')
if [ -z "$USER_POOL_ID" ]; then
  echo "ERROR: Failed to extract User Pool ID" >&2
  exit 1
fi

# Validate User Pool ID format
if ! [[ "$USER_POOL_ID" =~ ^[a-z]{2}-[a-z]+-[0-9]+_[a-zA-Z0-9]+$ ]]; then
  echo "ERROR: Invalid User Pool ID format: $USER_POOL_ID" >&2
  exit 1
fi

echo "User Pool created with ID: $USER_POOL_ID"

USER_POOL_ARN=$(echo "$USER_POOL_OUTPUT" | jq -r '.UserPool.Arn // empty')
aws cognito-idp tag-resource \
  --resource-arn "$USER_POOL_ARN" \
  --tags project=doc-smith,tutorial=amazon-cognito-gs

# Wait for user pool to be ready
echo "Waiting for user pool to be ready..."
sleep 5

# Step 2: Create an App Client with enhanced security
echo "Creating app client..."
APP_CLIENT_OUTPUT=$(aws cognito-idp create-user-pool-client \
  --user-pool-id "$USER_POOL_ID" \
  --client-name "$APP_CLIENT_NAME" \
  --no-generate-secret \
  --explicit-auth-flows ALLOW_REFRESH_TOKEN_AUTH ALLOW_USER_PASSWORD_AUTH \
  --callback-urls '["https://localhost:3000/callback"]' \
  --allowed-o-auth-flows 'code' \
  --allowed-o-auth-scopes 'openid' 'email' 'profile' \
  --allowed-o-auth-flows-user-pool-client \
```

```
  --prevent-user-existence-errors ENABLED \
  --enable-token-revocation \
  --access-token-validity 1 \
  --id-token-validity 1 \
  --refresh-token-validity 30 \
  --token-validity-units 'AccessToken=hours,IdToken=hours,RefreshToken=days' \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "create-user-pool-client"

# Extract the Client ID using jq for safety
CLIENT_ID=$(echo "$APP_CLIENT_OUTPUT" | jq -r '.UserPoolClient.ClientId //
 empty')
if [ -z "$CLIENT_ID" ]; then
  echo "ERROR: Failed to extract Client ID" >&2
  cleanup_on_error
  exit 1
fi

# Validate Client ID format
if ! [[ "$CLIENT_ID" =~ ^[a-z0-9]{26}$ ]]; then
  echo "ERROR: Invalid Client ID format: $CLIENT_ID" >&2
  cleanup_on_error
  exit 1
fi

echo "App Client created with ID: $CLIENT_ID"

# Step 3: Set Up a Domain for Your User Pool
echo "Setting up user pool domain..."
DOMAIN_OUTPUT=$(aws cognito-idp create-user-pool-domain \
  --user-pool-id "$USER_POOL_ID" \
  --domain "$DOMAIN_NAME" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "create-user-pool-domain"
echo "Domain created: $DOMAIN_NAME.auth.$AWS_REGION.amazoncognito.com"

# Step 4: View User Pool Details
echo "Retrieving user pool details..."
USER_POOL_DETAILS=$(aws cognito-idp describe-user-pool \
  --user-pool-id "$USER_POOL_ID" \
  --region "$AWS_REGION" \
  2>&1)
```

```
check_aws_error "describe-user-pool"
echo "User Pool details retrieved successfully"

# Step 5: View App Client Details
echo "Retrieving app client details..."
APP_CLIENT_DETAILS=$(aws cognito-idp describe-user-pool-client \
  --user-pool-id "$USER_POOL_ID" \
  --client-id "$CLIENT_ID" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "describe-user-pool-client"
echo "App Client details retrieved successfully"

# Step 6: Create a User (Admin)
echo "Creating admin user..."
ADMIN_USER_EMAIL="admin@example.com"
TEMP_PASSWORD="$(openssl rand -base64 12 | tr -d '\n')!@#"
if [ -z "$TEMP_PASSWORD" ]; then
  echo "ERROR: Failed to generate temporary password" >&2
  cleanup_on_error
  exit 1
fi

ADMIN_USER_OUTPUT=$(aws cognito-idp admin-create-user \
  --user-pool-id "$USER_POOL_ID" \
  --username "$ADMIN_USER_EMAIL" \
  --user-attributes Name=email,Value="$ADMIN_USER_EMAIL"
 Name=email_verified,Value=true \
  --temporary-password "$TEMP_PASSWORD" \
  --message-action SUPPRESS \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "admin-create-user"
echo "Admin user created: $ADMIN_USER_EMAIL"

# Securely clear temporary password from memory
unset TEMP_PASSWORD

# Step 7: Self-Registration
echo "Demonstrating self-registration..."
USER_EMAIL="user@example.com"
USER_PASSWORD="SecurePassword123!"
SIGNUP_OUTPUT=$(aws cognito-idp sign-up \
  --client-id "$CLIENT_ID" \
```

```
  --username "$USER_EMAIL" \
  --password "$USER_PASSWORD" \
  --user-attributes Name=email,Value="$USER_EMAIL" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "sign-up"
echo "User signed up: $USER_EMAIL"
echo "A confirmation code would be sent to the user's email in a real scenario"

# Securely clear password from memory
unset USER_PASSWORD

echo ""
echo "================================================="
echo "IMPORTANT: In a real scenario, the user would receive"
echo "a confirmation code via email. For this demo, we'll"
echo "use admin-confirm-sign-up instead."
echo "================================================="
echo ""

# Step 8: Confirm User Registration (using admin privileges for demo)
echo "Confirming user registration (admin method)..."
CONFIRM_OUTPUT=$(aws cognito-idp admin-confirm-sign-up \
  --user-pool-id "$USER_POOL_ID" \
  --username "$USER_EMAIL" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "admin-confirm-sign-up"
echo "User confirmed: $USER_EMAIL"

# Step 9: Set permanent password for user
echo "Setting permanent password for user..."
SET_PASSWORD="SecureUserPassword123!"
SET_PASS_OUTPUT=$(aws cognito-idp admin-set-user-password \
  --user-pool-id "$USER_POOL_ID" \
  --username "$USER_EMAIL" \
  --password "$SET_PASSWORD" \
  --permanent \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "admin-set-user-password"
echo "Permanent password set for user"

unset SET_PASSWORD
```

```
# Step 10: Authenticate a User
echo "Authenticating user..."
AUTH_PASSWORD="SecureUserPassword123!"
AUTH_OUTPUT=$(aws cognito-idp initiate-auth \
  --client-id "$CLIENT_ID" \
  --auth-flow USER_PASSWORD_AUTH \
  --auth-parameters "USERNAME=$USER_EMAIL,PASSWORD=$AUTH_PASSWORD" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "initiate-auth"
echo "User authenticated successfully"

unset AUTH_PASSWORD

# Extract auth tokens securely
ID_TOKEN=$(echo "$AUTH_OUTPUT" | jq -r '.AuthenticationResult.IdToken // empty')
ACCESS_TOKEN=$(echo "$AUTH_OUTPUT" | jq -r '.AuthenticationResult.AccessToken //
 empty')

# Validate tokens exist
if [ -z "$ID_TOKEN" ] || [ -z "$ACCESS_TOKEN" ]; then
  echo "WARNING: Failed to extract authentication tokens" >&2
else
  echo "Authentication tokens obtained successfully"
fi

# Securely clear tokens from memory
unset ID_TOKEN
unset ACCESS_TOKEN

# Step 11: List Users in the User Pool
echo "Listing users in the user pool..."
USERS_OUTPUT=$(aws cognito-idp list-users \
  --user-pool-id "$USER_POOL_ID" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "list-users"
echo "Users listed successfully"

# Display summary of created resources
echo ""
echo "================================================="
echo "RESOURCE SUMMARY"
```

```
echo "================================================="
echo "User Pool ID: $USER_POOL_ID"
echo "User Pool Name: $USER_POOL_NAME"
echo "App Client ID: $CLIENT_ID"
echo "App Client Name: $APP_CLIENT_NAME"
echo "Domain: $DOMAIN_NAME.auth.$AWS_REGION.amazoncognito.com"
echo "Admin User: $ADMIN_USER_EMAIL"
echo "Regular User: $USER_EMAIL"
echo "================================================="
echo ""

# Auto-confirm cleanup
echo ""
echo "========================================="
echo "CLEANUP"
echo "========================================="
echo "Starting cleanup process..."

# Step 12: Clean Up Resources
echo "Deleting user pool domain..."
DELETE_DOMAIN_OUTPUT=$(aws cognito-idp delete-user-pool-domain \
  --user-pool-id "$USER_POOL_ID" \
  --domain "$DOMAIN_NAME" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "delete-user-pool-domain"
echo "Domain deleted successfully"

# Wait for domain deletion to complete
echo "Waiting for domain deletion to complete..."
sleep 5

echo "Deleting user pool (this will also delete the app client)..."
DELETE_POOL_OUTPUT=$(aws cognito-idp delete-user-pool \
  --user-pool-id "$USER_POOL_ID" \
  --region "$AWS_REGION" \
  2>&1)
check_aws_error "delete-user-pool"
echo "User pool deleted successfully"

echo "All resources have been cleaned up"

echo "Script completed at $(date)"
```

```
# Remove trap to prevent cleanup on successful exit
trap - EXIT ERR
```

- For API details, see the following topics in *AWS CLI Command Reference*.
  - AdminConfirmSignUp
  - AdminCreateUser
  - CreateUserPool
  - CreateUserPoolClient
  - CreateUserPoolDomain
  - DeleteUserPool
  - DeleteUserPoolDomain
  - DescribeUserPool
  - DescribeUserPoolClient
  - InitiateAuth
  - ListUsers
  - SignUp

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Sign up a user with an Amazon Cognito user pool that requires MFA using an AWS SDK

The following code examples show how to:

- Sign up and confirm a user with a username, password, and email address.
- Set up multi-factor authentication by associating an MFA application with the user.
- Sign in by using a password and an MFA code.

.NET

SDK for .NET

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
namespace CognitoBasics;

public class CognitoBasics
{
    private static ILogger logger = null!;

    static async Task Main(string[] args)
    {
        // Set up dependency injection for Amazon Cognito.
        using var host = Host.CreateDefaultBuilder(args)
            .ConfigureLogging(logging =>
                logging.AddFilter("System", LogLevel.Debug)
                    .AddFilter<DebugLoggerProvider>("Microsoft",
 LogLevel.Information)
                    .AddFilter<ConsoleLoggerProvider>("Microsoft",
 LogLevel.Trace))
            .ConfigureServices((_, services) =>
            services.AddAWSService<IAmazonCognitoIdentityProvider>()
            .AddTransient<CognitoWrapper>()
            )
            .Build();

        logger = LoggerFactory.Create(builder => { builder.AddConsole(); })
            .CreateLogger<CognitoBasics>();

        var configuration = new ConfigurationBuilder()
            .SetBasePath(Directory.GetCurrentDirectory())
            .AddJsonFile("settings.json") // Load settings from .json file.
            .AddJsonFile("settings.local.json",
                true) // Optionally load local settings.
            .Build();
```

```
        var cognitoWrapper = host.Services.GetRequiredService<CognitoWrapper>();

        Console.WriteLine(new string('-', 80));
        UiMethods.DisplayOverview();
        Console.WriteLine(new string('-', 80));

        // clientId - The app client Id value that you get from the AWS CDK
script.
        var clientId = configuration["ClientId"]; // "*** REPLACE WITH CLIENT ID
VALUE FROM CDK SCRIPT";

        // poolId - The pool Id that you get from the AWS CDK script.
        var poolId = configuration["PoolId"]!; // "*** REPLACE WITH POOL ID VALUE
FROM CDK SCRIPT";
        var userName = configuration["UserName"];
        var password = configuration["Password"];
        var email = configuration["Email"];

        // If the username wasn't set in the configuration file,
        // get it from the user now.
        if (userName is null)
        {
            do
            {
                Console.Write("Username: ");
                userName = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(userName));
        }
        Console.WriteLine($"\nUsername: {userName}");

        // If the password wasn't set in the configuration file,
        // get it from the user now.
        if (password is null)
        {
            do
            {
                Console.Write("Password: ");
                password = Console.ReadLine();
            }
            while (string.IsNullOrEmpty(password));
        }

        // If the email address wasn't set in the configuration file,
```

```
        // get it from the user now.
        if (email is null)
        {
            do
            {
                Console.Write("Email: ");
                email = Console.ReadLine();
            } while (string.IsNullOrEmpty(email));
        }

        // Now sign up the user.
        Console.WriteLine($"\nSigning up {userName} with email address:
{email}");
        await cognitoWrapper.SignUpAsync(clientId, userName, password, email);

        // Add the user to the user pool.
        Console.WriteLine($"Adding {userName} to the user pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        UiMethods.DisplayTitle("Get confirmation code");
        Console.WriteLine($"Conformation code sent to {userName}.");
        Console.Write("Would you like to send a new code? (Y/N) ");
        var answer = Console.ReadLine();

        if (answer!.ToLower() == "y")
        {
            await cognitoWrapper.ResendConfirmationCodeAsync(clientId, userName);
            Console.WriteLine("Sending a new confirmation code");
        }

        Console.Write("Enter confirmation code (from Email): ");
        var code = Console.ReadLine();

        await cognitoWrapper.ConfirmSignupAsync(clientId, code, userName);

        UiMethods.DisplayTitle("Checking status");
        Console.WriteLine($"Rechecking the status of {userName} in the user
pool");
        await cognitoWrapper.GetAdminUserAsync(userName, poolId);

        Console.WriteLine($"Setting up authenticator for {userName} in the user
pool");
        var setupResponse = await cognitoWrapper.InitiateAuthAsync(clientId,
userName, password);
```

```
        var setupSession = await
 cognitoWrapper.AssociateSoftwareTokenAsync(setupResponse.Session);
        Console.Write("Enter the 6-digit code displayed in Google Authenticator:
 ");
        var setupCode = Console.ReadLine();

        var setupResult = await
 cognitoWrapper.VerifySoftwareTokenAsync(setupSession, setupCode);
        Console.WriteLine($"Setup status: {setupResult}");

        Console.WriteLine($"Now logging in {userName} in the user pool");
        var authSession = await cognitoWrapper.AdminInitiateAuthAsync(clientId,
 poolId, userName, password);

        Console.Write("Enter a new 6-digit code displayed in Google
 Authenticator: ");
        var authCode = Console.ReadLine();

        var authResult = await
 cognitoWrapper.AdminRespondToAuthChallengeAsync(userName, clientId, authCode,
 authSession, poolId);
        Console.WriteLine($"Authenticated and received access token:
 {authResult.AccessToken}");

        Console.WriteLine(new string('-', 80));
        Console.WriteLine("Cognito scenario is complete.");
        Console.WriteLine(new string('-', 80));
    }
}


using System.Net;

namespace CognitoActions;

/// <summary>
/// Methods to perform Amazon Cognito Identity Provider actions.
/// </summary>
public class CognitoWrapper
{
    private readonly IAmazonCognitoIdentityProvider _cognitoService;

    /// <summary>
```

```
    /// Constructor for the wrapper class containing Amazon Cognito actions.
    /// </summary>
    /// <param name="cognitoService">The Amazon Cognito client object.</param>
    public CognitoWrapper(IAmazonCognitoIdentityProvider cognitoService)
    {
        _cognitoService = cognitoService;
    }

    /// <summary>
    /// List the Amazon Cognito user pools for an account.
    /// </summary>
    /// <returns>A list of UserPoolDescriptionType objects.</returns>
    public async Task<List<UserPoolDescriptionType>> ListUserPoolsAsync()
    {
        var userPools = new List<UserPoolDescriptionType>();

        var userPoolsPaginator = _cognitoService.Paginators.ListUserPools(new
ListUserPoolsRequest());

        await foreach (var response in userPoolsPaginator.Responses)
        {
            userPools.AddRange(response.UserPools);
        }

        return userPools;
    }


    /// <summary>
    /// Get a list of users for the Amazon Cognito user pool.
    /// </summary>
    /// <param name="userPoolId">The user pool ID.</param>
    /// <returns>A list of users.</returns>
    public async Task<List<UserType>> ListUsersAsync(string userPoolId)
    {
        var request = new ListUsersRequest
        {
            UserPoolId = userPoolId
        };

        var users = new List<UserType>();

        var usersPaginator = _cognitoService.Paginators.ListUsers(request);
        await foreach (var response in usersPaginator.Responses)
```

```
        {
            users.AddRange(response.Users);
        }

        return users;
    }


    /// <summary>
    /// Respond to an admin authentication challenge.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="clientId">The client ID.</param>
    /// <param name="mfaCode">The multi-factor authentication code.</param>
    /// <param name="session">The current application session.</param>
    /// <param name="clientId">The user pool ID.</param>
    /// <returns>The result of the authentication response.</returns>
    public async Task<AuthenticationResultType> AdminRespondToAuthChallengeAsync(
        string userName,
        string clientId,
        string mfaCode,
        string session,
        string userPoolId)
    {
        Console.WriteLine("SOFTWARE_TOKEN_MFA challenge is generated");

        var challengeResponses = new Dictionary<string, string>();
        challengeResponses.Add("USERNAME", userName);
        challengeResponses.Add("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        var respondToAuthChallengeRequest = new
AdminRespondToAuthChallengeRequest
        {
            ChallengeName = ChallengeNameType.SOFTWARE_TOKEN_MFA,
            ClientId = clientId,
            ChallengeResponses = challengeResponses,
            Session = session,
            UserPoolId = userPoolId,
        };

        var response = await
_cognitoService.AdminRespondToAuthChallengeAsync(respondToAuthChallengeRequest);
        Console.WriteLine($"Response to Authentication
{response.AuthenticationResult.TokenType}");
```

```
        return response.AuthenticationResult;
    }


    /// <summary>
    /// Verify the TOTP and register for MFA.
    /// </summary>
    /// <param name="session">The name of the session.</param>
    /// <param name="code">The MFA code.</param>
    /// <returns>The status of the software token.</returns>
    public async Task<VerifySoftwareTokenResponseType>
VerifySoftwareTokenAsync(string session, string code)
    {
        var tokenRequest = new VerifySoftwareTokenRequest
        {
            UserCode = code,
            Session = session,
        };

        var verifyResponse = await
_cognitoService.VerifySoftwareTokenAsync(tokenRequest);

        return verifyResponse.Status;
    }


    /// <summary>
    /// Get an MFA token to authenticate the user with the authenticator.
    /// </summary>
    /// <param name="session">The session name.</param>
    /// <returns>The session name.</returns>
    public async Task<string> AssociateSoftwareTokenAsync(string session)
    {
        var softwareTokenRequest = new AssociateSoftwareTokenRequest
        {
            Session = session,
        };

        var tokenResponse = await
_cognitoService.AssociateSoftwareTokenAsync(softwareTokenRequest);
        var secretCode = tokenResponse.SecretCode;

        Console.WriteLine($"Use the following secret code to set up the
authenticator: {secretCode}");
```

```
        return tokenResponse.Session;
    }


    /// <summary>
    /// Initiate an admin auth request.
    /// </summary>
    /// <param name="clientId">The client ID to use.</param>
    /// <param name="userPoolId">The ID of the user pool.</param>
    /// <param name="userName">The username to authenticate.</param>
    /// <param name="password">The user's password.</param>
    /// <returns>The session to use in challenge-response.</returns>
    public async Task<string> AdminInitiateAuthAsync(string clientId, string
 userPoolId, string userName, string password)
    {
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var request = new AdminInitiateAuthRequest
        {
            ClientId = clientId,
            UserPoolId = userPoolId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.AdminInitiateAuthAsync(request);
        return response.Session;
    }

    /// <summary>
    /// Initiate authorization.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The name of the user who is authenticating.</
param>
    /// <param name="password">The password for the user who is authenticating.</
param>
    /// <returns>The response from the initiate auth request.</returns>
    public async Task<InitiateAuthResponse> InitiateAuthAsync(string clientId,
 string userName, string password)
    {
```

```
        var authParameters = new Dictionary<string, string>();
        authParameters.Add("USERNAME", userName);
        authParameters.Add("PASSWORD", password);

        var authRequest = new InitiateAuthRequest

        {
            ClientId = clientId,
            AuthParameters = authParameters,
            AuthFlow = AuthFlowType.USER_PASSWORD_AUTH,
        };

        var response = await _cognitoService.InitiateAuthAsync(authRequest);
        Console.WriteLine($"Result Challenge is : {response.ChallengeName}");

        return response;
    }

    /// <summary>
    /// Confirm that the user has signed up.
    /// </summary>
    /// <param name="clientId">The Id of this application.</param>
    /// <param name="code">The confirmation code sent to the user.</param>
    /// <param name="userName">The username.</param>
    /// <returns>True if successful.</returns>
    public async Task<bool> ConfirmSignupAsync(string clientId, string code,
string userName)
    {
        var signUpRequest = new ConfirmSignUpRequest
        {
            ClientId = clientId,
            ConfirmationCode = code,
            Username = userName,
        };

        var response = await _cognitoService.ConfirmSignUpAsync(signUpRequest);
        if (response.HttpStatusCode == HttpStatusCode.OK)
        {
            Console.WriteLine($"{userName} was confirmed");
            return true;
        }
        return false;
    }
```

```
    /// <summary>
    /// Initiates and confirms tracking of the device.
    /// </summary>
    /// <param name="accessToken">The user's access token.</param>
    /// <param name="deviceKey">The key of the device from Amazon Cognito.</
param>
    /// <param name="deviceName">The device name.</param>
    /// <returns></returns>
    public async Task<bool> ConfirmDeviceAsync(string accessToken, string
 deviceKey, string deviceName)
    {
        var request = new ConfirmDeviceRequest
        {
            AccessToken = accessToken,
            DeviceKey = deviceKey,
            DeviceName = deviceName
        };

        var response = await _cognitoService.ConfirmDeviceAsync(request);
        return response.UserConfirmationNecessary;
    }


    /// <summary>
    /// Send a new confirmation code to a user.
    /// </summary>
    /// <param name="clientId">The Id of the client application.</param>
    /// <param name="userName">The username of user who will receive the code.</
param>
    /// <returns>The delivery details.</returns>
    public async Task<CodeDeliveryDetailsType> ResendConfirmationCodeAsync(string
 clientId, string userName)
    {
        var codeRequest = new ResendConfirmationCodeRequest
        {
            ClientId = clientId,
            Username = userName,
        };

        var response = await
 _cognitoService.ResendConfirmationCodeAsync(codeRequest);
```

```
        Console.WriteLine($"Method of delivery is
 {response.CodeDeliveryDetails.DeliveryMedium}");

        return response.CodeDeliveryDetails;
    }


    /// <summary>
    /// Get the specified user from an Amazon Cognito user pool with
 administrator access.
    /// </summary>
    /// <param name="userName">The name of the user.</param>
    /// <param name="poolId">The Id of the Amazon Cognito user pool.</param>
    /// <returns>Async task.</returns>
    public async Task<UserStatusType> GetAdminUserAsync(string userName, string
 poolId)
    {
        AdminGetUserRequest userRequest = new AdminGetUserRequest
        {
            Username = userName,
            UserPoolId = poolId,
        };

        var response = await _cognitoService.AdminGetUserAsync(userRequest);

        Console.WriteLine($"User status {response.UserStatus}");
        return response.UserStatus;
    }


    /// <summary>
    /// Sign up a new user.
    /// </summary>
    /// <param name="clientId">The client Id of the application.</param>
    /// <param name="userName">The username to use.</param>
    /// <param name="password">The user's password.</param>
    /// <param name="email">The email address of the user.</param>
    /// <returns>A Boolean value indicating whether the user was confirmed.</
returns>
    public async Task<bool> SignUpAsync(string clientId, string userName, string
 password, string email)
    {
        var userAttrs = new AttributeType
        {
```

```
            Name = "email",
            Value = email,
        };

        var userAttrsList = new List<AttributeType>();

        userAttrsList.Add(userAttrs);

        var signUpRequest = new SignUpRequest
        {
            UserAttributes = userAttrsList,
            Username = userName,
            ClientId = clientId,
            Password = password
        };

        var response = await _cognitoService.SignUpAsync(signUpRequest);
        return response.HttpStatusCode == HttpStatusCode.OK;
    }

}
```

- For API details, see the following topics in *AWS SDK for .NET API Reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

C++

SDK for C++

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
        Aws::Client::ClientConfiguration clientConfig;
        // Optional: Set to the AWS Region (overrides config file).
        // clientConfig.region = "us-east-1";

//! Scenario that adds a user to an Amazon Cognito user pool.
/*!
  \sa gettingStartedWithUserPools()
  \param clientID: Client ID associated with an Amazon Cognito user pool.
  \param userPoolID: An Amazon Cognito user pool ID.
  \param clientConfig: Aws client configuration.
  \return bool: Successful completion.
 */
bool AwsDoc::Cognito::gettingStartedWithUserPools(const Aws::String &clientID,
                                                  const Aws::String &userPoolID,
                                                  const
 Aws::Client::ClientConfiguration &clientConfig) {
    printAsterisksLine();
    std::cout
            << "Welcome to the Amazon Cognito example scenario."
            << std::endl;
    printAsterisksLine();

    std::cout
            << "This scenario will add a user to an Amazon Cognito user pool."
            << std::endl;
    const Aws::String userName = askQuestion("Enter a new username: ");
    const Aws::String password = askQuestion("Enter a new password: ");
    const Aws::String email = askQuestion("Enter a valid email for the user: ");

    std::cout << "Signing up " << userName << std::endl;
```

```
   Aws::CognitoIdentityProvider::CognitoIdentityProviderClient
client(clientConfig);
   bool userExists = false;
   do {
       // 1. Add a user with a username, password, and email address.
       Aws::CognitoIdentityProvider::Model::SignUpRequest request;
       request.AddUserAttributes(
               Aws::CognitoIdentityProvider::Model::AttributeType().WithName(
                       "email").WithValue(email));
       request.SetUsername(userName);
       request.SetPassword(password);
       request.SetClientId(clientID);
       Aws::CognitoIdentityProvider::Model::SignUpOutcome outcome =
               client.SignUp(request);

       if (outcome.IsSuccess()) {
           std::cout << "The signup request for " << userName << " was
successful."
                     << std::endl;
       }
       else if (outcome.GetError().GetErrorType() ==

Aws::CognitoIdentityProvider::CognitoIdentityProviderErrors::USERNAME_EXISTS) {
           std::cout
                   << "The username already exists. Please enter a different
username."
                   << std::endl;
           userExists = true;
       }
       else {
           std::cerr << "Error with CognitoIdentityProvider::SignUpRequest. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   } while (userExists);

   printAsterisksLine();
   std::cout << "Retrieving status of " << userName << " in the user pool."
             << std::endl;
   // 2. Confirm that the user was added to the user pool.
   if (!checkAdminUserStatus(userName, userPoolID, client)) {
       return false;
   }
```

```
   std::cout << "A confirmation code was sent to " << email << "." << std::endl;

   bool resend = askYesNoQuestion("Would you like to send a new code? (y/n) ");
   if (resend) {
       // Request a resend of the confirmation code to the email address.
(ResendConfirmationCode)
       Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeRequest
request;
       request.SetUsername(userName);
       request.SetClientId(clientID);

       Aws::CognitoIdentityProvider::Model::ResendConfirmationCodeOutcome
outcome =
               client.ResendConfirmationCode(request);

       if (outcome.IsSuccess()) {
           std::cout
                   << "CognitoIdentityProvider::ResendConfirmationCode was
successful."
                   << std::endl;
       }
       else {
           std::cerr << "Error with
CognitoIdentityProvider::ResendConfirmationCode. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }
   }

   printAsterisksLine();

   {
       // 4. Send the confirmation code that's received in the email.
(ConfirmSignUp)
       const Aws::String confirmationCode = askQuestion(
               "Enter the confirmation code that was emailed: ");
       Aws::CognitoIdentityProvider::Model::ConfirmSignUpRequest request;
       request.SetClientId(clientID);
       request.SetConfirmationCode(confirmationCode);
       request.SetUsername(userName);

       Aws::CognitoIdentityProvider::Model::ConfirmSignUpOutcome outcome =
```

```
                client.ConfirmSignUp(request);

        if (outcome.IsSuccess()) {
            std::cout << "ConfirmSignup was Successful."
                      << std::endl;
        }
        else {
            std::cerr << "Error with CognitoIdentityProvider::ConfirmSignUp. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    std::cout << "Rechecking the status of " << userName << " in the user pool."
              << std::endl;
    if (!checkAdminUserStatus(userName, userPoolID, client)) {
        return false;
    }

    printAsterisksLine();

    std::cout << "Initiating authorization using the username and password."
              << std::endl;

    Aws::String session;
    // 5. Initiate authorization with username and password. (AdminInitiateAuth)
    if (!adminInitiateAuthorization(clientID, userPoolID,  userName, password,
 session, client)) {
        return false;
    }

    printAsterisksLine();

    std::cout
            << "Starting setup of time-based one-time password (TOTP) multi-
factor authentication (MFA)."
            << std::endl;

    {
        // 6. Request a setup key for one-time password (TOTP)
        //    multi-factor authentication (MFA). (AssociateSoftwareToken)
        Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenRequest
 request;
```

```
        request.SetSession(session);

        Aws::CognitoIdentityProvider::Model::AssociateSoftwareTokenOutcome
 outcome =
                client.AssociateSoftwareToken(request);

        if (outcome.IsSuccess()) {
            std::cout
                    << "Enter this setup key into an authenticator app, for
 example Google Authenticator."
                    << std::endl;
            std::cout << "Setup key: " << outcome.GetResult().GetSecretCode()
                      << std::endl;
#ifdef USING_QR
            printAsterisksLine();
            std::cout << "\nOr scan the QR code in the file '" << QR_CODE_PATH <<
 "."
                      << std::endl;

            saveQRCode(std::string("otpauth://totp/") + userName + "?secret=" +
                       outcome.GetResult().GetSecretCode());
#endif // USING_QR
            session = outcome.GetResult().GetSession();
        }
        else {
            std::cerr << "Error with
 CognitoIdentityProvider::AssociateSoftwareToken. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }
    askQuestion("Type enter to continue...", alwaysTrueTest);

    printAsterisksLine();

    {
        Aws::String userCode = askQuestion(
                "Enter the 6 digit code displayed in the authenticator app: ");

        //  7. Send the MFA code copied from an authenticator app.
 (VerifySoftwareToken)
        Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenRequest request;
        request.SetUserCode(userCode);
```

```
        request.SetSession(session);

        Aws::CognitoIdentityProvider::Model::VerifySoftwareTokenOutcome outcome =
                client.VerifySoftwareToken(request);

        if (outcome.IsSuccess()) {
            std::cout << "Verification of the code was successful."
                      << std::endl;
            session = outcome.GetResult().GetSession();
        }
        else {
            std::cerr << "Error with
CognitoIdentityProvider::VerifySoftwareToken. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
            return false;
        }
    }

    printAsterisksLine();
    std::cout << "You have completed the MFA authentication setup." << std::endl;
    std::cout << "Now, sign in." << std::endl;

    // 8. Initiate authorization again with username and password.
(AdminInitiateAuth)
    if (!adminInitiateAuthorization(clientID, userPoolID, userName, password,
session, client)) {
        return false;
    }

    Aws::String accessToken;
    {
        Aws::String mfaCode = askQuestion(
                "Re-enter the 6 digit code displayed in the authenticator app:
");

        // 9. Send a new MFA code copied from an authenticator app.
(AdminRespondToAuthChallenge)
        Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeRequest
request;
        request.AddChallengeResponses("USERNAME", userName);
        request.AddChallengeResponses("SOFTWARE_TOKEN_MFA_CODE", mfaCode);
        request.SetChallengeName(
```

```
Aws::CognitoIdentityProvider::Model::ChallengeNameType::SOFTWARE_TOKEN_MFA);
       request.SetClientId(clientID);
       request.SetUserPoolId(userPoolID);
       request.SetSession(session);

       Aws::CognitoIdentityProvider::Model::AdminRespondToAuthChallengeOutcome
outcome =
               client.AdminRespondToAuthChallenge(request);

       if (outcome.IsSuccess()) {
           std::cout << "Here is the response to the challenge.\n" <<

outcome.GetResult().GetAuthenticationResult().Jsonize().View().WriteReadable()
                     << std::endl;

           accessToken =
outcome.GetResult().GetAuthenticationResult().GetAccessToken();
       }
       else {
           std::cerr << "Error with
CognitoIdentityProvider::AdminRespondToAuthChallenge. "
                     << outcome.GetError().GetMessage()
                     << std::endl;
           return false;
       }

       std::cout << "You have successfully added a user to Amazon Cognito."
                 << std::endl;
   }

   if (askYesNoQuestion("Would you like to delete the user that you just added?
(y/n) ")) {
       // 10. Delete the user that you just added. (DeleteUser)
       Aws::CognitoIdentityProvider::Model::DeleteUserRequest request;
       request.SetAccessToken(accessToken);

       Aws::CognitoIdentityProvider::Model::DeleteUserOutcome outcome =
               client.DeleteUser(request);

       if (outcome.IsSuccess()) {
           std::cout << "The user " << userName << " was deleted."
                     << std::endl;
       }
```

```
        else {
            std::cerr << "Error with CognitoIdentityProvider::DeleteUser. "
                      << outcome.GetError().GetMessage()
                      << std::endl;
        }
    }

    return true;
}

//! Routine which checks the user status in an Amazon Cognito user pool.
/*!
 \sa checkAdminUserStatus()
 \param userName: A username.
 \param userPoolID: An Amazon Cognito user pool ID.
 \return bool: Successful completion.
 */
bool AwsDoc::Cognito::checkAdminUserStatus(const Aws::String &userName,
                                           const Aws::String &userPoolID,
                                           const
 Aws::CognitoIdentityProvider::CognitoIdentityProviderClient &client) {
    Aws::CognitoIdentityProvider::Model::AdminGetUserRequest request;
    request.SetUsername(userName);
    request.SetUserPoolId(userPoolID);

    Aws::CognitoIdentityProvider::Model::AdminGetUserOutcome outcome =
            client.AdminGetUser(request);

    if (outcome.IsSuccess()) {
        std::cout << "The status for " << userName << " is " <<

 Aws::CognitoIdentityProvider::Model::UserStatusTypeMapper::GetNameForUserStatusType(
                          outcome.GetResult().GetUserStatus()) << std::endl;
        std::cout << "Enabled is " << outcome.GetResult().GetEnabled() <<
 std::endl;
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::AdminGetUser. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

```
//! Routine which starts authorization of an Amazon Cognito user.
//! This routine requires administrator credentials.
/*!
 \sa adminInitiateAuthorization()
 \param clientID: Client ID of tracked device.
 \param userPoolID: An Amazon Cognito user pool ID.
 \param userName: A username.
 \param password: A password.
 \param sessionResult: String to receive a session token.
 \return bool: Successful completion.
 */
bool AwsDoc::Cognito::adminInitiateAuthorization(const Aws::String &clientID,
                                                 const Aws::String &userPoolID,
                                                 const Aws::String &userName,
                                                 const Aws::String &password,
                                                 Aws::String &sessionResult,
                                                 const
 Aws::CognitoIdentityProvider::CognitoIdentityProviderClient &client) {
    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthRequest request;
    request.SetClientId(clientID);
    request.SetUserPoolId(userPoolID);
    request.AddAuthParameters("USERNAME", userName);
    request.AddAuthParameters("PASSWORD", password);
    request.SetAuthFlow(

 Aws::CognitoIdentityProvider::Model::AuthFlowType::ADMIN_USER_PASSWORD_AUTH);


    Aws::CognitoIdentityProvider::Model::AdminInitiateAuthOutcome outcome =
            client.AdminInitiateAuth(request);

    if (outcome.IsSuccess()) {
        std::cout << "Call to AdminInitiateAuth was successful." << std::endl;
        sessionResult = outcome.GetResult().GetSession();
    }
    else {
        std::cerr << "Error with CognitoIdentityProvider::AdminInitiateAuth. "
                  << outcome.GetError().GetMessage()
                  << std::endl;
    }

    return outcome.IsSuccess();
}
```

- For API details, see the following topics in *AWS SDK for C++ API Reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Java

SDK for Java 2.x

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
import software.amazon.awssdk.regions.Region;
import
 software.amazon.awssdk.services.cognitoidentityprovider.CognitoIdentityProviderClient;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminGetUserRequest;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminGetUserResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminInitiateAuthRequest;
```

```
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminInitiateAuthResponse;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChalleng
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AdminRespondToAuthChalleng
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenRequ
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AssociateSoftwareTokenResp
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AttributeType;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.AuthFlowType;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ChallengeNameType;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.CognitoIdentityProviderExc
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ConfirmSignUpRequest;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeRequ
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.ResendConfirmationCodeResp
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.SignUpRequest;
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenRequest
import
 software.amazon.awssdk.services.cognitoidentityprovider.model.VerifySoftwareTokenRespons
import java.security.InvalidKeyException;
import java.security.NoSuchAlgorithmException;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Scanner;

/**
 * Before running this Java V2 code example, set up your development
 * environment, including your credentials.
 *
 * For more information, see the following documentation:
 *
```

```
 * https://docs.aws.amazon.com/sdk-for-java/latest/developer-guide/get-
started.html
 *
 * TIP: To set up the required user pool, run the AWS Cloud Development Kit (AWS
 * CDK) script provided in this GitHub repo at
 * resources/cdk/cognito_scenario_user_pool_with_mfa.
 *
 * This code example performs the following operations:
 *
 * 1. Invokes the signUp method to sign up a user.
 * 2. Invokes the adminGetUser method to get the user's confirmation status.
 * 3. Invokes the ResendConfirmationCode method if the user requested another
 * code.
 * 4. Invokes the confirmSignUp method.
 * 5. Invokes the AdminInitiateAuth to sign in. This results in being prompted
 * to set up TOTP (time-based one-time password). (The response is
 * “ChallengeName”: “MFA_SETUP”).
 * 6. Invokes the AssociateSoftwareToken method to generate a TOTP MFA private
 * key. This can be used with Google Authenticator.
 * 7. Invokes the VerifySoftwareToken method to verify the TOTP and register for
 * MFA.
 * 8. Invokes the AdminInitiateAuth to sign in again. This results in being
 * prompted to submit a TOTP (Response: “ChallengeName”: “SOFTWARE_TOKEN_MFA”).
 * 9. Invokes the AdminRespondToAuthChallenge to get back a token.
 */

public class CognitoMVP {
    public static final String DASHES = new String(new char[80]).replace("\0",
 "-");

    public static void main(String[] args) throws NoSuchAlgorithmException,
 InvalidKeyException {
        final String usage = """

                Usage:
                    <clientId> <poolId>

                Where:
                    clientId - The app client Id value that you can get from the
 AWS CDK script.
                    poolId - The pool Id that you can get from the AWS CDK
 script.\s
                """;
```

```
        if (args.length != 2) {
            System.out.println(usage);
            System.exit(1);
        }

        String clientId = args[0];
        String poolId = args[1];
        CognitoIdentityProviderClient identityProviderClient =
CognitoIdentityProviderClient.builder()
                .region(Region.US_EAST_1)
                .build();

        System.out.println(DASHES);
        System.out.println("Welcome to the Amazon Cognito example scenario.");
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Enter your user name");
        Scanner in = new Scanner(System.in);
        String userName = in.nextLine();

        System.out.println("*** Enter your password");
        String password = in.nextLine();

        System.out.println("*** Enter your email");
        String email = in.nextLine();

        System.out.println("1. Signing up " + userName);
        signUp(identityProviderClient, clientId, userName, password, email);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("2. Getting " + userName + " in the user pool");
        getAdminUser(identityProviderClient, userName, poolId);

        System.out
                .println("*** Conformation code sent to " + userName + ". Would
you like to send a new code? (Yes/No)");
        System.out.println(DASHES);

        System.out.println(DASHES);
        String ans = in.nextLine();

        if (ans.compareTo("Yes") == 0) {
```

```
            resendConfirmationCode(identityProviderClient, clientId, userName);
            System.out.println("3. Sending a new confirmation code");
        }
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("4. Enter confirmation code that was emailed");
        String code = in.nextLine();
        confirmSignUp(identityProviderClient, clientId, code, userName);
        System.out.println("Rechecking the status of " + userName + " in the user
pool");
        getAdminUser(identityProviderClient, userName, poolId);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("5. Invokes the initiateAuth to sign in");
        AdminInitiateAuthResponse authResponse =
initiateAuth(identityProviderClient, clientId, userName, password,
                poolId);
        String mySession = authResponse.session();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("6. Invokes the AssociateSoftwareToken method to
generate a TOTP key");
        String newSession = getSecretForAppMFA(identityProviderClient,
mySession);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("*** Enter the 6-digit code displayed in Google
Authenticator");
        String myCode = in.nextLine();
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("7. Verify the TOTP and register for MFA");
        verifyTOTP(identityProviderClient, newSession, myCode);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("8. Re-enter a 6-digit code displayed in Google
Authenticator");
        String mfaCode = in.nextLine();
```

```
        AdminInitiateAuthResponse authResponse1 =
initiateAuth(identityProviderClient, clientId, userName, password,
                poolId);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("9.  Invokes the AdminRespondToAuthChallenge");
        String session2 = authResponse1.session();
        adminRespondToAuthChallenge(identityProviderClient, userName, clientId,
mfaCode, session2);
        System.out.println(DASHES);

        System.out.println(DASHES);
        System.out.println("All Amazon Cognito operations were successfully
performed");
        System.out.println(DASHES);
    }

    // Respond to an authentication challenge.
    public static void adminRespondToAuthChallenge(CognitoIdentityProviderClient
identityProviderClient,
            String userName, String clientId, String mfaCode, String session) {
        System.out.println("SOFTWARE_TOKEN_MFA challenge is generated");
        Map<String, String> challengeResponses = new HashMap<>();

        challengeResponses.put("USERNAME", userName);
        challengeResponses.put("SOFTWARE_TOKEN_MFA_CODE", mfaCode);

        AdminRespondToAuthChallengeRequest respondToAuthChallengeRequest =
AdminRespondToAuthChallengeRequest.builder()
                .challengeName(ChallengeNameType.SOFTWARE_TOKEN_MFA)
                .clientId(clientId)
                .challengeResponses(challengeResponses)
                .session(session)
                .build();

        AdminRespondToAuthChallengeResponse respondToAuthChallengeResult =
identityProviderClient
                .adminRespondToAuthChallenge(respondToAuthChallengeRequest);

System.out.println("respondToAuthChallengeResult.getAuthenticationResult()"
                + respondToAuthChallengeResult.authenticationResult());
    }
```

```
   // Verify the TOTP and register for MFA.
   public static void verifyTOTP(CognitoIdentityProviderClient
identityProviderClient, String session, String code) {
       try {
           VerifySoftwareTokenRequest tokenRequest =
VerifySoftwareTokenRequest.builder()
                   .userCode(code)
                   .session(session)
                   .build();

           VerifySoftwareTokenResponse verifyResponse =
identityProviderClient.verifySoftwareToken(tokenRequest);
           System.out.println("The status of the token is " +
verifyResponse.statusAsString());

       } catch (CognitoIdentityProviderException e) {
           System.err.println(e.awsErrorDetails().errorMessage());
           System.exit(1);
       }
   }

   public static AdminInitiateAuthResponse
initiateAuth(CognitoIdentityProviderClient identityProviderClient,
           String clientId, String userName, String password, String userPoolId)
{
       try {
           Map<String, String> authParameters = new HashMap<>();
           authParameters.put("USERNAME", userName);
           authParameters.put("PASSWORD", password);

           AdminInitiateAuthRequest authRequest =
AdminInitiateAuthRequest.builder()
                   .clientId(clientId)
                   .userPoolId(userPoolId)
                   .authParameters(authParameters)
                   .authFlow(AuthFlowType.ADMIN_USER_PASSWORD_AUTH)
                   .build();

           AdminInitiateAuthResponse response =
identityProviderClient.adminInitiateAuth(authRequest);
           System.out.println("Result Challenge is : " +
response.challengeName());
           return response;
```

```
        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }

        return null;
    }

    public static String getSecretForAppMFA(CognitoIdentityProviderClient
identityProviderClient, String session) {
        AssociateSoftwareTokenRequest softwareTokenRequest =
AssociateSoftwareTokenRequest.builder()
                .session(session)
                .build();

        AssociateSoftwareTokenResponse tokenResponse = identityProviderClient
                .associateSoftwareToken(softwareTokenRequest);
        String secretCode = tokenResponse.secretCode();
        System.out.println("Enter this token into Google Authenticator");
        System.out.println(secretCode);
        return tokenResponse.session();
    }

    public static void confirmSignUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String code,
            String userName) {
        try {
            ConfirmSignUpRequest signUpRequest = ConfirmSignUpRequest.builder()
                    .clientId(clientId)
                    .confirmationCode(code)
                    .username(userName)
                    .build();

            identityProviderClient.confirmSignUp(signUpRequest);
            System.out.println(userName + " was confirmed");

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void resendConfirmationCode(CognitoIdentityProviderClient
identityProviderClient, String clientId,
```

```
            String userName) {
        try {
            ResendConfirmationCodeRequest codeRequest =
ResendConfirmationCodeRequest.builder()
                    .clientId(clientId)
                    .username(userName)
                    .build();

            ResendConfirmationCodeResponse response =
identityProviderClient.resendConfirmationCode(codeRequest);
            System.out.println("Method of delivery is " +
response.codeDeliveryDetails().deliveryMediumAsString());

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }

    public static void signUp(CognitoIdentityProviderClient
identityProviderClient, String clientId, String userName,
            String password, String email) {
        AttributeType userAttrs = AttributeType.builder()
                .name("email")
                .value(email)
                .build();

        List<AttributeType> userAttrsList = new ArrayList<>();
        userAttrsList.add(userAttrs);
        try {
            SignUpRequest signUpRequest = SignUpRequest.builder()
                    .userAttributes(userAttrsList)
                    .username(userName)
                    .clientId(clientId)
                    .password(password)
                    .build();

            identityProviderClient.signUp(signUpRequest);
            System.out.println("User has been signed up ");

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
```

```
    }

    public static void getAdminUser(CognitoIdentityProviderClient
 identityProviderClient, String userName,
            String poolId) {
        try {
            AdminGetUserRequest userRequest = AdminGetUserRequest.builder()
                    .username(userName)
                    .userPoolId(poolId)
                    .build();

            AdminGetUserResponse response =
 identityProviderClient.adminGetUser(userRequest);
            System.out.println("User status " + response.userStatusAsString());

        } catch (CognitoIdentityProviderException e) {
            System.err.println(e.awsErrorDetails().errorMessage());
            System.exit(1);
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Java 2.x API Reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

JavaScript

SDK for JavaScript (v3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

For the best experience, clone the GitHub repository and run this example. The following code represents a sample of the full example application.

```
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { signUp } from "../../../actions/sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateUser = (username, password, email) => {
  if (!(username && password && email)) {
    throw new Error(
      `Username, password, and email must be provided as arguments to the 'sign-
up' command.`,
    );
  }
};

const signUpHandler = async (commands) => {
  const [_, username, password, email] = commands;

  try {
    validateUser(username, password, email);
    /**
```

```
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Signing up.");
    await signUp({ clientId, username, password, email });
    logger.log(`Signed up. A confirmation email has been sent to: ${email}.`);
    logger.log(
      `Run 'confirm-sign-up ${username} <code>' to confirm your account.`,
    );
  } catch (err) {
    logger.error(err);
  }
};

export { signUpHandler };

const signUp = ({ clientId, username, password, email }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new SignUpCommand({
    ClientId: clientId,
    Username: username,
    Password: password,
    UserAttributes: [{ Name: "email", Value: email }],
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { confirmSignUp } from "../../../actions/confirm-sign-up.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getSecondValuesFromEntries } from "@aws-doc-sdk-examples/lib/utils/util-
csv.js";

const validateClient = (clientId) => {
  if (!clientId) {
    throw new Error(
      `App client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};
```

```
const validateUser = (username) => {
  if (!username) {
    throw new Error(
      `Username name is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const validateCode = (code) => {
  if (!code) {
    throw new Error(
      `Verification code is missing. It must be provided as an argument to the
 'confirm-sign-up' command.`,
    );
  }
};

const confirmSignUpHandler = async (commands) => {
  const [_, username, code] = commands;

  try {
    validateUser(username);
    validateCode(code);
    /**
     * @type {string[]}
     */
    const values = getSecondValuesFromEntries(FILE_USER_POOLS);
    const clientId = values[0];
    validateClient(clientId);
    logger.log("Confirming user.");
    await confirmSignUp({ clientId, username, code });
    logger.log(
      `User confirmed. Run 'admin-initiate-auth ${username} <password>' to sign
 in.`,
    );
  } catch (err) {
    logger.error(err);
  }
};

export { confirmSignUpHandler };
```

```
const confirmSignUp = ({ clientId, username, code }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new ConfirmSignUpCommand({
    ClientId: clientId,
    Username: username,
    ConfirmationCode: code,
  });

  return client.send(command);
};

import qrcode from "qrcode-terminal";
import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminInitiateAuth } from "../../../actions/admin-initiate-auth.js";
import { associateSoftwareToken } from "../../../actions/associate-software-
token.js";
import { FILE_USER_POOLS } from "./constants.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";

const handleMfaSetup = async (session, username) => {
  const { SecretCode, Session } = await associateSoftwareToken(session);

  // Store the Session for use with 'VerifySoftwareToken'.
  process.env.SESSION = Session;

  console.log(
    "Scan this code in your preferred authenticator app, then run 'verify-
software-token' to finish the setup.",
  );
  qrcode.generate(
    `otpauth://totp/${username}?secret=${SecretCode}`,
    { small: true },
    console.log,
  );
};

const handleSoftwareTokenMfa = (session) => {
  // Store the Session for use with 'AdminRespondToAuthChallenge'.
  process.env.SESSION = session;
};

const validateClient = (id) => {
  if (!id) {
```

```
    throw new Error(
      `User pool client id is missing. Did you run 'create-user-pool'?`,
    );
  }
};

const validateId = (id) => {
  if (!id) {
    throw new Error(`User pool id is missing. Did you run 'create-user-pool'?`);
  }
};

const validateUser = (username, password) => {
  if (!(username && password)) {
    throw new Error(
      `Username and password must be provided as arguments to the 'admin-
initiate-auth' command.`,
    );
  }
};

const adminInitiateAuthHandler = async (commands) => {
  const [_, username, password] = commands;

  try {
    validateUser(username, password);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    validateId(userPoolId);
    validateClient(clientId);

    logger.log("Signing in.");
    const { ChallengeName, Session } = await adminInitiateAuth({
      clientId,
      userPoolId,
      username,
      password,
    });

    if (ChallengeName === "MFA_SETUP") {
      logger.log("MFA setup is required.");
      return handleMfaSetup(Session, username);
    }
```

```
    if (ChallengeName === "SOFTWARE_TOKEN_MFA") {
      handleSoftwareTokenMfa(Session);
      logger.log(`Run 'admin-respond-to-auth-challenge ${username} <totp>'`);
    }
  } catch (err) {
    logger.error(err);
  }
};

export { adminInitiateAuthHandler };

const adminInitiateAuth = ({ clientId, userPoolId, username, password }) => {
  const client = new CognitoIdentityProviderClient({});

  const command = new AdminInitiateAuthCommand({
    ClientId: clientId,
    UserPoolId: userPoolId,
    AuthFlow: AuthFlowType.ADMIN_USER_PASSWORD_AUTH,
    AuthParameters: { USERNAME: username, PASSWORD: password },
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { adminRespondToAuthChallenge } from "../../../actions/admin-respond-to-
auth-challenge.js";
import { getFirstEntry } from "@aws-doc-sdk-examples/lib/utils/util-csv.js";
import { FILE_USER_POOLS } from "./constants.js";

const verifyUsername = (username) => {
  if (!username) {
    throw new Error(
      `Username is missing. It must be provided as an argument to the 'admin-
respond-to-auth-challenge' command.`,
    );
  }
};

const verifyTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) is missing. It must be provided as an
 argument to the 'admin-respond-to-auth-challenge' command.`,
```

```
    );
  }
};

const storeAccessToken = (token) => {
  process.env.AccessToken = token;
};

const adminRespondToAuthChallengeHandler = async (commands) => {
  const [_, username, totp] = commands;

  try {
    verifyUsername(username);
    verifyTotp(totp);

    const [userPoolId, clientId] = getFirstEntry(FILE_USER_POOLS);
    const session = process.env.SESSION;

    const { AuthenticationResult } = await adminRespondToAuthChallenge({
      clientId,
      userPoolId,
      username,
      totp,
      session,
    });

    storeAccessToken(AuthenticationResult.AccessToken);

    logger.log("Successfully authenticated.");
  } catch (err) {
    logger.error(err);
  }
};

export { adminRespondToAuthChallengeHandler };

const respondToAuthChallenge = ({
  clientId,
  username,
  session,
  userPoolId,
  code,
}) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  const command = new RespondToAuthChallengeCommand({
    ChallengeName: ChallengeNameType.SOFTWARE_TOKEN_MFA,
    ChallengeResponses: {
      SOFTWARE_TOKEN_MFA_CODE: code,
      USERNAME: username,
    },
    ClientId: clientId,
    UserPoolId: userPoolId,
    Session: session,
  });

  return client.send(command);
};

import { logger } from "@aws-doc-sdk-examples/lib/utils/util-log.js";
import { verifySoftwareToken } from "../../../actions/verify-software-token.js";

const validateTotp = (totp) => {
  if (!totp) {
    throw new Error(
      `Time-based one-time password (TOTP) must be provided to the 'validate-
software-token' command.`,
    );
  }
};
const verifySoftwareTokenHandler = async (commands) => {
  const [_, totp] = commands;

  try {
    validateTotp(totp);

    logger.log("Verifying TOTP.");
    await verifySoftwareToken(totp);
    logger.log("TOTP Verified. Run 'admin-initiate-auth' again to sign-in.");
  } catch (err) {
    logger.error(err);
  }
};

export { verifySoftwareTokenHandler };

const verifySoftwareToken = (totp) => {
  const client = new CognitoIdentityProviderClient({});
```

```
  // The 'Session' is provided in the response to 'AssociateSoftwareToken'.
  const session = process.env.SESSION;

  if (!session) {
    throw new Error(
      "Missing a valid Session. Did you run 'admin-initiate-auth'?",
    );
  }

  const command = new VerifySoftwareTokenCommand({
    Session: session,
    UserCode: totp,
  });

  return client.send(command);
};
```

- For API details, see the following topics in *AWS SDK for JavaScript API Reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Kotlin

SDK for Kotlin

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
/**
 Before running this Kotlin code example, set up your development environment,
 including your credentials.

 For more information, see the following documentation:
 https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html

 TIP: To set up the required user pool, run the AWS Cloud Development
 Kit (AWS CDK) script provided in this GitHub repo at resources/cdk/
cognito_scenario_user_pool_with_mfa.

 This code example performs the following operations:

 1. Invokes the signUp method to sign up a user.
 2. Invokes the adminGetUser method to get the user's confirmation status.
 3. Invokes the ResendConfirmationCode method if the user requested another code.
 4. Invokes the confirmSignUp method.
 5. Invokes the initiateAuth to sign in. This results in being prompted to
 set up TOTP (time-based one-time password). (The response is “ChallengeName”:
 “MFA_SETUP”).
 6. Invokes the AssociateSoftwareToken method to generate a TOTP MFA private key.
 This can be used with Google Authenticator.
 7. Invokes the VerifySoftwareToken method to verify the TOTP and register for
 MFA.
 8. Invokes the AdminInitiateAuth to sign in again. This results in being
 prompted to submit a TOTP (Response: “ChallengeName”: “SOFTWARE_TOKEN_MFA”).
 9. Invokes the AdminRespondToAuthChallenge to get back a token.
 */

suspend fun main(args: Array<String>) {
    val usage = """
```

```
        Usage:
            <clientId> <poolId>
        Where:
            clientId - The app client Id value that you can get from the AWS CDK
script.
            poolId - The pool Id that you can get from the AWS CDK script.
    """

    if (args.size != 2) {
        println(usage)
        exitProcess(1)
    }

    val clientId = args[0]
    val poolId = args[1]

    // Use the console to get data from the user.
    println("*** Enter your use name")
    val inOb = Scanner(System.`in`)
    val userName = inOb.nextLine()
    println(userName)

    println("*** Enter your password")
    val password: String = inOb.nextLine()

    println("*** Enter your email")
    val email = inOb.nextLine()

    println("*** Signing up $userName")
    signUp(clientId, userName, password, email)

    println("*** Getting $userName in the user pool")
    getAdminUser(userName, poolId)

    println("*** Conformation code sent to $userName. Would you like to send a
new code? (Yes/No)")
    val ans = inOb.nextLine()

    if (ans.compareTo("Yes") == 0) {
        println("*** Sending a new confirmation code")
        resendConfirmationCode(clientId, userName)
    }
    println("*** Enter the confirmation code that was emailed")
    val code = inOb.nextLine()
```

```
    confirmSignUp(clientId, code, userName)

    println("*** Rechecking the status of $userName in the user pool")
    getAdminUser(userName, poolId)

    val authResponse = checkAuthMethod(clientId, userName, password, poolId)
    val mySession = authResponse.session
    val newSession = getSecretForAppMFA(mySession)
    println("*** Enter the 6-digit code displayed in Google Authenticator")
    val myCode = inOb.nextLine()

    // Verify the TOTP and register for MFA.
    verifyTOTP(newSession, myCode)
    println("*** Re-enter a 6-digit code displayed in Google Authenticator")
    val mfaCode: String = inOb.nextLine()
    val authResponse1 = checkAuthMethod(clientId, userName, password, poolId)
    val session2 = authResponse1.session
    adminRespondToAuthChallenge(userName, clientId, mfaCode, session2)
}

suspend fun checkAuthMethod(
    clientIdVal: String,
    userNameVal: String,
    passwordVal: String,
    userPoolIdVal: String,
): AdminInitiateAuthResponse {
    val authParas = mutableMapOf<String, String>()
    authParas["USERNAME"] = userNameVal
    authParas["PASSWORD"] = passwordVal

    val authRequest =
        AdminInitiateAuthRequest {
            clientId = clientIdVal
            userPoolId = userPoolIdVal
            authParameters = authParas
            authFlow = AuthFlowType.AdminUserPasswordAuth
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.adminInitiateAuth(authRequest)
        println("Result Challenge is ${response.challengeName}")
        return response
    }
```

```
}

suspend fun resendConfirmationCode(
    clientIdVal: String?,
    userNameVal: String?,
) {
    val codeRequest =
        ResendConfirmationCodeRequest {
            clientId = clientIdVal
            username = userNameVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.resendConfirmationCode(codeRequest)
        println("Method of delivery is " +
 (response.codeDeliveryDetails?.deliveryMedium))
    }
}

// Respond to an authentication challenge.
suspend fun adminRespondToAuthChallenge(
    userName: String,
    clientIdVal: String?,
    mfaCode: String,
    sessionVal: String?,
) {
    println("SOFTWARE_TOKEN_MFA challenge is generated")
    val challengeResponsesOb = mutableMapOf<String, String>()
    challengeResponsesOb["USERNAME"] = userName
    challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

    val adminRespondToAuthChallengeRequest =
        AdminRespondToAuthChallengeRequest {
            challengeName = ChallengeNameType.SoftwareTokenMfa
            clientId = clientIdVal
            challengeResponses = challengeResponsesOb
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val respondToAuthChallengeResult =
 identityProviderClient.adminRespondToAuthChallenge(adminRespondToAuthChallengeRequest)
```

```
        println("respondToAuthChallengeResult.getAuthenticationResult()
 ${respondToAuthChallengeResult.authenticationResult}")
    }
}

// Verify the TOTP and register for MFA.
suspend fun verifyTOTP(
    sessionVal: String?,
    codeVal: String?,
) {
    val tokenRequest =
        VerifySoftwareTokenRequest {
            userCode = codeVal
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val verifyResponse =
 identityProviderClient.verifySoftwareToken(tokenRequest)
        println("The status of the token is ${verifyResponse.status}")
    }
}

suspend fun getSecretForAppMFA(sessionVal: String?): String? {
    val softwareTokenRequest =
        AssociateSoftwareTokenRequest {
            session = sessionVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val tokenResponse =
 identityProviderClient.associateSoftwareToken(softwareTokenRequest)
        val secretCode = tokenResponse.secretCode
        println("Enter this token into Google Authenticator")
        println(secretCode)
        return tokenResponse.session
    }
}

suspend fun confirmSignUp(
    clientIdVal: String?,
    codeVal: String?,
```

```
    userNameVal: String?,
) {
    val signUpRequest =
        ConfirmSignUpRequest {
            clientId = clientIdVal
            confirmationCode = codeVal
            username = userNameVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        identityProviderClient.confirmSignUp(signUpRequest)
        println("$userNameVal  was confirmed")
    }
}

suspend fun getAdminUser(
    userNameVal: String?,
    poolIdVal: String?,
) {
    val userRequest =
        AdminGetUserRequest {
            username = userNameVal
            userPoolId = poolIdVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        val response = identityProviderClient.adminGetUser(userRequest)
        println("User status ${response.userStatus}")
    }
}

suspend fun signUp(
    clientIdVal: String?,
    userNameVal: String?,
    passwordVal: String?,
    emailVal: String?,
) {
    val userAttrs =
        AttributeType {
            name = "email"
            value = emailVal
        }
```

```
    val userAttrsList = mutableListOf<AttributeType>()
    userAttrsList.add(userAttrs)
    val signUpRequest =
        SignUpRequest {
            userAttributes = userAttrsList
            username = userNameVal
            clientId = clientIdVal
            password = passwordVal
        }

    CognitoIdentityProviderClient.fromEnvironment { region = "us-east-1" }.use
 { identityProviderClient ->
        identityProviderClient.signUp(signUpRequest)
        println("User has been signed up")
    }
}
```

- For API details, see the following topics in *AWS SDK for Kotlin API reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp
  - VerifySoftwareToken

Python

SDK for Python (Boto3)

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Create a class that wraps Amazon Cognito functions used in the scenario.

```
class CognitoIdentityProviderWrapper:
    """Encapsulates Amazon Cognito actions"""

    def __init__(self, cognito_idp_client, user_pool_id, client_id,
 client_secret=None):
        """
        :param cognito_idp_client: A Boto3 Amazon Cognito Identity Provider
 client.
        :param user_pool_id: The ID of an existing Amazon Cognito user pool.
        :param client_id: The ID of a client application registered with the user
 pool.
        :param client_secret: The client secret, if the client has a secret.
        """
        self.cognito_idp_client = cognito_idp_client
        self.user_pool_id = user_pool_id
        self.client_id = client_id
        self.client_secret = client_secret


    def _secret_hash(self, user_name):
        """
        Calculates a secret hash from a user name and a client secret.

        :param user_name: The user name to use when calculating the hash.
        :return: The secret hash.
        """
        key = self.client_secret.encode()
        msg = bytes(user_name + self.client_id, "utf-8")
        secret_hash = base64.b64encode(
            hmac.new(key, msg, digestmod=hashlib.sha256).digest()
        ).decode()
```

```
        logger.info("Made secret hash for %s: %s.", user_name, secret_hash)
        return secret_hash

    def sign_up_user(self, user_name, password, user_email):
        """
        Signs up a new user with Amazon Cognito. This action prompts Amazon
Cognito
        to send an email to the specified email address. The email contains a
code that
        can be used to confirm the user.

        When the user already exists, the user status is checked to determine
whether
        the user has been confirmed.

        :param user_name: The user name that identifies the new user.
        :param password: The password for the new user.
        :param user_email: The email address for the new user.
        :return: True when the user is already confirmed with Amazon Cognito.
                 Otherwise, false.
        """
        try:
            kwargs = {
                "ClientId": self.client_id,
                "Username": user_name,
                "Password": password,
                "UserAttributes": [{"Name": "email", "Value": user_email}],
            }
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            response = self.cognito_idp_client.sign_up(**kwargs)
            confirmed = response["UserConfirmed"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "UsernameExistsException":
                response = self.cognito_idp_client.admin_get_user(
                    UserPoolId=self.user_pool_id, Username=user_name
                )
                logger.warning(
                    "User %s exists and is %s.", user_name,
response["UserStatus"]
                )
                confirmed = response["UserStatus"] == "CONFIRMED"
            else:
                logger.error(
```

```
                    "Couldn't sign up %s. Here's why: %s: %s",
                    user_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        return confirmed


    def resend_confirmation(self, user_name):
        """
        Prompts Amazon Cognito to resend an email with a new confirmation code.

        :param user_name: The name of the user who will receive the email.
        :return: Delivery information about where the email is sent.
        """
        try:
            kwargs = {"ClientId": self.client_id, "Username": user_name}
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            response = self.cognito_idp_client.resend_confirmation_code(**kwargs)
            delivery = response["CodeDeliveryDetails"]
        except ClientError as err:
            logger.error(
                "Couldn't resend confirmation to %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return delivery


    def confirm_user_sign_up(self, user_name, confirmation_code):
        """
        Confirms a previously created user. A user must be confirmed before they
        can sign in to Amazon Cognito.

        :param user_name: The name of the user to confirm.
        :param confirmation_code: The confirmation code sent to the user's
registered
                                  email address.
        :return: True when the confirmation succeeds.
```

```
        """
        try:
            kwargs = {
                "ClientId": self.client_id,
                "Username": user_name,
                "ConfirmationCode": confirmation_code,
            }
            if self.client_secret is not None:
                kwargs["SecretHash"] = self._secret_hash(user_name)
            self.cognito_idp_client.confirm_sign_up(**kwargs)
        except ClientError as err:
            logger.error(
                "Couldn't confirm sign up for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return True


    def list_users(self):
        """
        Returns a list of the users in the current user pool.

        :return: The list of users.
        """
        try:
            response =
self.cognito_idp_client.list_users(UserPoolId=self.user_pool_id)
            users = response["Users"]
        except ClientError as err:
            logger.error(
                "Couldn't list users for %s. Here's why: %s: %s",
                self.user_pool_id,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return users
```

```
    def start_sign_in(self, user_name, password):
        """
        Starts the sign-in process for a user by using administrator credentials.
        This method of signing in is appropriate for code running on a secure
 server.

        If the user pool is configured to require MFA and this is the first sign-
in
        for the user, Amazon Cognito returns a challenge response to set up an
        MFA application. When this occurs, this function gets an MFA secret from
        Amazon Cognito and returns it to the caller.

        :param user_name: The name of the user to sign in.
        :param password: The user's password.
        :return: The result of the sign-in attempt. When sign-in is successful,
 this
                 returns an access token that can be used to get AWS credentials.
 Otherwise,
                 Amazon Cognito returns a challenge to set up an MFA application,
                 or a challenge to enter an MFA code from a registered MFA
 application.
        """
        try:
            kwargs = {
                "UserPoolId": self.user_pool_id,
                "ClientId": self.client_id,
                "AuthFlow": "ADMIN_USER_PASSWORD_AUTH",
                "AuthParameters": {"USERNAME": user_name, "PASSWORD": password},
            }
            if self.client_secret is not None:
                kwargs["AuthParameters"]["SECRET_HASH"] =
 self._secret_hash(user_name)
            response = self.cognito_idp_client.admin_initiate_auth(**kwargs)
            challenge_name = response.get("ChallengeName", None)
            if challenge_name == "MFA_SETUP":
                if (
                    "SOFTWARE_TOKEN_MFA"
                    in response["ChallengeParameters"]["MFAS_CAN_SETUP"]
                ):
                    response.update(self.get_mfa_secret(response["Session"]))
                else:
                    raise RuntimeError(
                        "The user pool requires MFA setup, but the user pool is
 not "
```

```
                        "configured for TOTP MFA. This example requires TOTP
MFA."
                    )
        except ClientError as err:
            logger.error(
                "Couldn't start sign in for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response


    def get_mfa_secret(self, session):
        """
        Gets a token that can be used to associate an MFA application with the
user.

        :param session: Session information returned from a previous call to
initiate
                        authentication.
        :return: An MFA token that can be used to set up an MFA application.
        """
        try:
            response =
self.cognito_idp_client.associate_software_token(Session=session)
        except ClientError as err:
            logger.error(
                "Couldn't get MFA secret. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response


    def verify_mfa(self, session, user_code):
        """
        Verify a new MFA application that is associated with a user.
```

```
        :param session: Session information returned from a previous call to
initiate
                        authentication.
        :param user_code: A code generated by the associated MFA application.
        :return: Status that indicates whether the MFA application is verified.
        """
        try:
            response = self.cognito_idp_client.verify_software_token(
                Session=session, UserCode=user_code
            )
        except ClientError as err:
            logger.error(
                "Couldn't verify MFA. Here's why: %s: %s",
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            response.pop("ResponseMetadata", None)
            return response


    def respond_to_mfa_challenge(self, user_name, session, mfa_code):
        """
        Responds to a challenge for an MFA code. This completes the second step
of
        a two-factor sign-in. When sign-in is successful, it returns an access
token
        that can be used to get AWS credentials from Amazon Cognito.

        :param user_name: The name of the user who is signing in.
        :param session: Session information returned from a previous call to
initiate
                        authentication.
        :param mfa_code: A code generated by the associated MFA application.
        :return: The result of the authentication. When successful, this contains
an
                 access token for the user.
        """
        try:
            kwargs = {
                "UserPoolId": self.user_pool_id,
                "ClientId": self.client_id,
```

```
                "ChallengeName": "SOFTWARE_TOKEN_MFA",
                "Session": session,
                "ChallengeResponses": {
                    "USERNAME": user_name,
                    "SOFTWARE_TOKEN_MFA_CODE": mfa_code,
                },
            }
            if self.client_secret is not None:
                kwargs["ChallengeResponses"]["SECRET_HASH"] = self._secret_hash(
                    user_name
                )
            response =
self.cognito_idp_client.admin_respond_to_auth_challenge(**kwargs)
            auth_result = response["AuthenticationResult"]
        except ClientError as err:
            if err.response["Error"]["Code"] == "ExpiredCodeException":
                logger.warning(
                    "Your MFA code has expired or has been used already. You
might have "
                    "to wait a few seconds until your app shows you a new code."
                )
            else:
                logger.error(
                    "Couldn't respond to mfa challenge for %s. Here's why: %s:
%s",
                    user_name,
                    err.response["Error"]["Code"],
                    err.response["Error"]["Message"],
                )
                raise
        else:
            return auth_result


    def confirm_mfa_device(
        self,
        user_name,
        device_key,
        device_group_key,
        device_password,
        access_token,
        aws_srp,
    ):
        """
```

```
        Confirms an MFA device to be tracked by Amazon Cognito. When a device is
        tracked, its key and password can be used to sign in without requiring a
new
        MFA code from the MFA application.

        :param user_name: The user that is associated with the device.
        :param device_key: The key of the device, returned by Amazon Cognito.
        :param device_group_key: The group key of the device, returned by Amazon
Cognito.
        :param device_password: The password that is associated with the device.
        :param access_token: The user's access token.
        :param aws_srp: A class that helps with Secure Remote Password (SRP)
                        calculations. The scenario associated with this example
uses
                        the warrant package.
        :return: True when the user must confirm the device. Otherwise, False.
When
                 False, the device is automatically confirmed and tracked.
        """
        srp_helper = aws_srp.AWSSRP(
            username=user_name,
            password=device_password,
            pool_id="_",
            client_id=self.client_id,
            client_secret=None,
            client=self.cognito_idp_client,
        )
        device_and_pw = f"{device_group_key}{device_key}:{device_password}"
        device_and_pw_hash = aws_srp.hash_sha256(device_and_pw.encode("utf-8"))
        salt = aws_srp.pad_hex(aws_srp.get_random(16))
        x_value = aws_srp.hex_to_long(aws_srp.hex_hash(salt +
device_and_pw_hash))
        verifier = aws_srp.pad_hex(pow(srp_helper.val_g, x_value,
srp_helper.big_n))
        device_secret_verifier_config = {
            "PasswordVerifier": base64.standard_b64encode(
                bytearray.fromhex(verifier)
            ).decode("utf-8"),
            "Salt":
base64.standard_b64encode(bytearray.fromhex(salt)).decode("utf-8"),
        }
        try:
            response = self.cognito_idp_client.confirm_device(
                AccessToken=access_token,
```

```
                DeviceKey=device_key,
                DeviceSecretVerifierConfig=device_secret_verifier_config,
            )
            user_confirm = response["UserConfirmationNecessary"]
        except ClientError as err:
            logger.error(
                "Couldn't confirm mfa device %s. Here's why: %s: %s",
                device_key,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return user_confirm


    def sign_in_with_tracked_device(
        self,
        user_name,
        password,
        device_key,
        device_group_key,
        device_password,
        aws_srp,
    ):
        """
        Signs in to Amazon Cognito as a user who has a tracked device. Signing in
        with a tracked device lets a user sign in without entering a new MFA
 code.

        Signing in with a tracked device requires that the client respond to the
 SRP
        protocol. The scenario associated with this example uses the warrant
 package
        to help with SRP calculations.

        For more information on SRP, see https://en.wikipedia.org/wiki/
Secure_Remote_Password_protocol.

        :param user_name: The user that is associated with the device.
        :param password: The user's password.
        :param device_key: The key of a tracked device.
        :param device_group_key: The group key of a tracked device.
        :param device_password: The password that is associated with the device.
```

```
        :param aws_srp: A class that helps with SRP calculations. The scenario
                        associated with this example uses the warrant package.
        :return: The result of the authentication. When successful, this contains
an
                 access token for the user.
        """
        try:
            srp_helper = aws_srp.AWSSRP(
                username=user_name,
                password=device_password,
                pool_id="_",
                client_id=self.client_id,
                client_secret=None,
                client=self.cognito_idp_client,
            )

            response_init = self.cognito_idp_client.initiate_auth(
                ClientId=self.client_id,
                AuthFlow="USER_PASSWORD_AUTH",
                AuthParameters={
                    "USERNAME": user_name,
                    "PASSWORD": password,
                    "DEVICE_KEY": device_key,
                },
            )
            if response_init["ChallengeName"] != "DEVICE_SRP_AUTH":
                raise RuntimeError(
                    f"Expected DEVICE_SRP_AUTH challenge but got
{response_init['ChallengeName']}."
                )

            auth_params = srp_helper.get_auth_params()
            auth_params["DEVICE_KEY"] = device_key
            response_auth = self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_SRP_AUTH",
                ChallengeResponses=auth_params,
            )
            if response_auth["ChallengeName"] != "DEVICE_PASSWORD_VERIFIER":
                raise RuntimeError(
                    f"Expected DEVICE_PASSWORD_VERIFIER challenge but got "
                    f"{response_init['ChallengeName']}."
                )
```

```
            challenge_params = response_auth["ChallengeParameters"]
            challenge_params["USER_ID_FOR_SRP"] = device_group_key + device_key
            cr = srp_helper.process_challenge(challenge_params, {"USERNAME":
 user_name})
            cr["USERNAME"] = user_name
            cr["DEVICE_KEY"] = device_key
            response_verifier =
 self.cognito_idp_client.respond_to_auth_challenge(
                ClientId=self.client_id,
                ChallengeName="DEVICE_PASSWORD_VERIFIER",
                ChallengeResponses=cr,
            )
            auth_tokens = response_verifier["AuthenticationResult"]
        except ClientError as err:
            logger.error(
                "Couldn't start client sign in for %s. Here's why: %s: %s",
                user_name,
                err.response["Error"]["Code"],
                err.response["Error"]["Message"],
            )
            raise
        else:
            return auth_tokens
```

Create a class that runs the scenario. This example also registers an MFA device to be tracked by Amazon Cognito and shows you how to sign in by using a password and information from the tracked device. This avoids the need to enter a new MFA code.

```
def run_scenario(cognito_idp_client, user_pool_id, client_id):
    logging.basicConfig(level=logging.INFO, format="%(levelname)s: %(message)s")

    print("-" * 88)
    print("Welcome to the Amazon Cognito user signup with MFA demo.")
    print("-" * 88)

    cog_wrapper = CognitoIdentityProviderWrapper(
        cognito_idp_client, user_pool_id, client_id
    )
```

```
    user_name = q.ask("Let's sign up a new user. Enter a user name: ",
q.non_empty)
    password = q.ask("Enter a password for the user: ", q.non_empty)
    email = q.ask("Enter a valid email address that you own: ", q.non_empty)
    confirmed = cog_wrapper.sign_up_user(user_name, password, email)
    while not confirmed:
        print(
            f"User {user_name} requires confirmation. Check {email} for "
            f"a verification code."
        )
        confirmation_code = q.ask("Enter the confirmation code from the email: ")
        if not confirmation_code:
            if q.ask("Do you need another confirmation code (y/n)? ",
q.is_yesno):
                delivery = cog_wrapper.resend_confirmation(user_name)
                print(
                    f"Confirmation code sent by {delivery['DeliveryMedium']} "
                    f"to {delivery['Destination']}."
                )
        else:
            confirmed = cog_wrapper.confirm_user_sign_up(user_name,
confirmation_code)
    print(f"User {user_name} is confirmed and ready to use.")
    print("-" * 88)

    print("Let's get a list of users in the user pool.")
    q.ask("Press Enter when you're ready.")
    users = cog_wrapper.list_users()
    if users:
        print(f"Found {len(users)} users:")
        pp(users)
    else:
        print("No users found.")
    print("-" * 88)

    print("Let's sign in and get an access token.")
    auth_tokens = None
    challenge = "ADMIN_USER_PASSWORD_AUTH"
    response = {}
    while challenge is not None:
        if challenge == "ADMIN_USER_PASSWORD_AUTH":
            response = cog_wrapper.start_sign_in(user_name, password)
            # start_sign_in may return an AuthenticationResult with no challenge
            # (for example, when the user pool does not require MFA). Read the
```

```
            # challenge defensively and stop if sign-in already succeeded.
            challenge = response.get("ChallengeName")
            if challenge is None:
                auth_tokens = response.get("AuthenticationResult")
                if auth_tokens is not None:
                    print(f"You're signed in as {user_name}.")
                    print("Here's your access token:")
                    pp(auth_tokens["AccessToken"])
                break
        elif response["ChallengeName"] == "MFA_SETUP":
            print("First, we need to set up an MFA application.")
            qr_img = qrcode.make(
                f"otpauth://totp/{user_name}?secret={response['SecretCode']}"
            )
            qr_img.save("qr.png")
            q.ask(
                "Press Enter to see a QR code on your screen. Scan it into an MFA
"
                "application, such as Google Authenticator."
            )
            webbrowser.open("qr.png")
            mfa_code = q.ask(
                "Enter the verification code from your MFA application: ",
q.non_empty
            )
            response = cog_wrapper.verify_mfa(response["Session"], mfa_code)
            print(f"MFA device setup {response['Status']}")
            print("Now that an MFA application is set up, let's sign in again.")
            print(
                "You might have to wait a few seconds for a new MFA code to
appear in "
                "your MFA application."
            )
            challenge = "ADMIN_USER_PASSWORD_AUTH"
        elif response["ChallengeName"] == "SOFTWARE_TOKEN_MFA":
            auth_tokens = None
            while auth_tokens is None:
                mfa_code = q.ask(
                    "Enter a verification code from your MFA application: ",
q.non_empty
                )
                auth_tokens = cog_wrapper.respond_to_mfa_challenge(
                    user_name, response["Session"], mfa_code
                )
```

```
            print(f"You're signed in as {user_name}.")
            print("Here's your access token:")
            pp(auth_tokens["AccessToken"])
            if "NewDeviceMetadata" in auth_tokens:
                print("And your device information:")
                pp(auth_tokens["NewDeviceMetadata"])
            challenge = None
        else:
            raise Exception(f"Got unexpected challenge
 {response['ChallengeName']}")
    print("-" * 88)

    # If sign-in returned neither a challenge nor an AuthenticationResult, there
    # are no tokens to work with; nothing more to do.
    if auth_tokens is None:
        return user_name

    # respond_to_mfa_challenge returns NewDeviceMetadata only when the user pool
    # has device tracking configured. When it's absent, skip the device-
confirmation
    # and tracked-device sign-in steps instead of failing on a missing key.
    device_metadata = auth_tokens.get("NewDeviceMetadata")
    if device_metadata is None:
        print(
            "This user pool doesn't track devices, so there's no device to
 confirm. "
            "To try the tracked-device sign-in flow, enable device tracking on
 the "
            "user pool and run this example again."
        )
        print("-" * 88)
    else:
        device_group_key = device_metadata["DeviceGroupKey"]
        device_key = device_metadata["DeviceKey"]
        device_password =
 base64.standard_b64encode(os.urandom(40)).decode("utf-8")

        print(
            "Let's confirm your MFA device so you don't have to re-enter MFA
 tokens for it."
        )
        q.ask("Press Enter when you're ready.")
        cog_wrapper.confirm_mfa_device(
            user_name,
```

```
            device_key,
            device_group_key,
            device_password,
            auth_tokens["AccessToken"],
            aws_srp,
        )
        print(f"Your device {device_key} is confirmed.")
        print("-" * 88)

        print(
            f"Now let's sign in as {user_name} from your confirmed device
 {device_key}.\n"
            f"Because this device is tracked by Amazon Cognito, you won't have to
 re-enter an MFA code."
        )
        q.ask("Press Enter when ready.")
        auth_tokens = cog_wrapper.sign_in_with_tracked_device(
            user_name, password, device_key, device_group_key, device_password,
 aws_srp
        )
        print("You're signed in. Your access token is:")
        pp(auth_tokens["AccessToken"])
        print("-" * 88)

    print("Don't forget to delete your user pool when you're done with this
 example.")
    print("\nThanks for watching!")
    print("-" * 88)
    return user_name


def main():
    parser = argparse.ArgumentParser(
        description="Shows how to sign up a new user with Amazon Cognito and
 associate "
        "the user with an MFA application for multi-factor authentication."
    )
    parser.add_argument(
        "user_pool_id", help="The ID of the user pool to use for the example."
    )
    parser.add_argument(
        "client_id", help="The ID of the client application to use for the
 example."
    )
```

```
    args = parser.parse_args()
    client = boto3.client("cognito-idp")
    created_user = None
    try:
        created_user = run_scenario(client, args.user_pool_id, args.client_id)
    except Exception:
        logging.exception("Something went wrong with the demo.")
    finally:
        # Clean up the user this scenario created, on success or failure, so a
        # run never leaves an orphaned user in the pool. Cleanup errors are
        # logged but do not mask the original outcome.
        if created_user is not None:
            try:
                client.admin_delete_user(
                    UserPoolId=args.user_pool_id, Username=created_user
                )
                print(f"Deleted user {created_user}.")
            except Exception:
                logging.exception(
                    "Couldn't delete user %s; delete it manually.", created_user
                )


if __name__ == "__main__":
    main()
```

- For API details, see the following topics in *AWS SDK for Python (Boto3) API Reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge
  - AssociateSoftwareToken
  - ConfirmDevice
  - ConfirmSignUp
  - InitiateAuth
  - ListUsers
  - ResendConfirmationCode
  - RespondToAuthChallenge
  - SignUp

- VerifySoftwareToken

Swift

SDK for Swift

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

The `Package.swift` file.

```
// swift-tools-version: 5.9
//
// The swift-tools-version declares the minimum version of Swift required to
// build this package.

import PackageDescription

let package = Package(
    name: "cognito-scenario",
    // Let Xcode know the minimum Apple platforms supported.
    platforms: [
        .macOS(.v13),
        .iOS(.v15)
    ],
    dependencies: [
        // Dependencies declare other packages that this package depends on.
        .package(
            url: "https://github.com/awslabs/aws-sdk-swift",
            from: "1.0.0"),
        .package(
            url: "https://github.com/apple/swift-argument-parser.git",
            branch: "main"
        )
    ],
    targets: [
        // Targets are the basic building blocks of a package, defining a module
 or a test suite.
        // Targets can depend on other targets in this package and products
```

```
        // from dependencies.
        .executableTarget(
            name: "cognito-scenario",
            dependencies: [
                .product(name: "AWSCognitoIdentityProvider", package: "aws-sdk-
swift"),
                .product(name: "ArgumentParser", package: "swift-argument-
parser")
            ],
            path: "Sources")

    ]
)
```

The Swift code file.

```
// An example demonstrating various features of Amazon Cognito. Before running
// this Swift code example, set up your development environment, including
// your credentials.
//
// For more information, see the following documentation:
// https://docs.aws.amazon.com/sdk-for-kotlin/latest/developer-guide/setup.html
//
// TIP: To set up the required user pool, run the AWS Cloud Development Kit
// (AWS CDK) script provided in this GitHub repo at
// resources/cdk/cognito_scenario_user_pool_with_mfa.
//
// This example performs the following functions:
//
// 1. Invokes the signUp method to sign up a user.
// 2. Invokes the adminGetUser method to get the user's confirmation status.
// 3. Invokes the ResendConfirmationCode method if the user requested another
//    code.
// 4. Invokes the confirmSignUp method.
// 5. Invokes the initiateAuth to sign in. This results in being prompted to
//    set up TOTP (time-based one-time password). (The response is
//    “ChallengeName”: “MFA_SETUP”).
// 6. Invokes the AssociateSoftwareToken method to generate a TOTP MFA private
//    key. This can be used with Google Authenticator.
// 7. Invokes the VerifySoftwareToken method to verify the TOTP and register
//    for MFA.
// 8. Invokes the AdminInitiateAuth to sign in again. This results in being
```

```
//    prompted to submit a TOTP (Response: “ChallengeName”:
//    “SOFTWARE_TOKEN_MFA”).
// 9. Invokes the AdminRespondToAuthChallenge to get back a token.

import ArgumentParser
import Foundation

import AWSClientRuntime
import AWSCognitoIdentityProvider

struct ExampleCommand: ParsableCommand {
    @Argument(help: "The application clientId.")
    var clientId: String
    @Argument(help: "The user pool ID to use.")
    var poolId: String
    @Option(help: "Name of the Amazon Region to use")
    var region = "us-east-1"

    static var configuration = CommandConfiguration(
        commandName: "cognito-scenario",
        abstract: """
        Demonstrates various features of Amazon Cognito.
        """,
        discussion: """
        """
    )

    /// Prompt for an input string of at least a minimum length.
    ///
    /// - Parameters:
    ///   - prompt: The prompt string to display.
    ///   - minLength: The minimum number of characters to allow in the
    ///     response. Default value is 0.
    ///
    /// - Returns: The entered string.
    func stringRequest(_ prompt: String, minLength: Int = 1) -> String {
        while true {
            print(prompt, terminator: "")
            let str = readLine()

            guard let str else {
                continue
            }
            if str.count >= minLength {
```

```
                return str
            } else {
                print("*** Response must be at least \(minLength) character(s)
long.")
            }
        }
    }

    /// Ask a yes/no question.
    ///
    /// - Parameter prompt: A prompt string to print.
    ///
    /// - Returns: `true` if the user answered "Y", otherwise `false`.
    func yesNoRequest(_ prompt: String) -> Bool {
        while true {
            let answer = stringRequest(prompt).lowercased()
            if answer == "y" || answer == "n" {
                return answer == "y"
            }
        }
    }

    /// Get information about a specific user in a user pool.
    ///
    /// - Parameters:
    ///   - cipClient: The Amazon Cognito Identity Provider client to use.
    ///   - userName: The user to retrieve information about.
    ///   - userPoolId: The user pool to search for the specified user.
    ///
    /// - Returns: `true` if the user's information was successfully
    ///   retrieved. Otherwise returns `false`.
    func adminGetUser(cipClient: CognitoIdentityProviderClient, userName: String,
                      userPoolId: String) async -> Bool {
        do {
            let output = try await cipClient.adminGetUser(
                input: AdminGetUserInput(
                    userPoolId: userPoolId,
                    username: userName
                )
            )

            guard let userStatus = output.userStatus else {
                print("*** Unable to get the user's status.")
                return false
```

```
            }

            print("User status: \(userStatus)")
            return true
        } catch {
            return false
        }
    }

    /// Create a new user in a user pool.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The ID of the app client to create a user for.
    ///   - userName: The username for the new user.
    ///   - password: The new user's password.
    ///   - email: The new user's email address.
    ///
    /// - Returns: `true` if successful; otherwise `false`.
    func signUp(cipClient: CognitoIdentityProviderClient, clientId: String,
userName: String, password: String, email: String) async -> Bool {
        let emailAttr = CognitoIdentityProviderClientTypes.AttributeType(
            name: "email",
            value: email
        )

        let userAttrsList = [emailAttr]

        do {
            _ = try await cipClient.signUp(
                input: SignUpInput(
                    clientId: clientId,
                    password: password,
                    userAttributes: userAttrsList,
                    username: userName
                )

            )

            print("=====> User \(userName) signed up.")
        } catch _ as AWSCognitoIdentityProvider.UsernameExistsException {
            print("*** The username \(userName) already exists. Please use a
different one.")
            return false
```

```
        } catch let error as AWSCognitoIdentityProvider.InvalidPasswordException
{
            print("*** Error: The specified password is invalid. Reason:
\(error.properties.message ?? "<none available>").")
            return false
        } catch _ as AWSCognitoIdentityProvider.ResourceNotFoundException {
            print("*** Error: The specified client ID (\(clientId)) doesn't
exist.")
            return false
        } catch {
            print("*** Unexpected error: \(error)")
            return false
        }

        return true
    }

    /// Requests a new confirmation code be sent to the given user's contact
    /// method.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The application client ID.
    ///   - userName: The user to resend a code for.
    ///
    /// - Returns: `true` if a new code was sent successfully, otherwise
    ///   `false`.
    func resendConfirmationCode(cipClient: CognitoIdentityProviderClient,
clientId: String,
                                userName: String) async -> Bool {
        do {
            let output = try await cipClient.resendConfirmationCode(
                input: ResendConfirmationCodeInput(
                    clientId: clientId,
                    username: userName
                )
            )

            guard let deliveryMedium = output.codeDeliveryDetails?.deliveryMedium
else {
                print("*** Unable to get the delivery method for the resent
code.")
                return false
            }
```

```
            print("=====> A new code has been sent by \(deliveryMedium)")
            return true
        } catch {
            print("*** Unable to resend the confirmation code to user
\(userName).")
            return false
        }
    }

    /// Submit a confirmation code for the specified user. This is the code as
    /// entered by the user after they've received it by email or text
    /// message.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - clientId: The app client ID the user is signing up for.
    ///   - userName: The username of the user whose code is being sent.
    ///   - code: The user's confirmation code.
    ///
    /// - Returns: `true` if the code was successfully confirmed; otherwise
`false`.
    func confirmSignUp(cipClient: CognitoIdentityProviderClient, clientId:
String,
                       userName: String, code: String) async -> Bool {
        do {
            _ = try await cipClient.confirmSignUp(
                input: ConfirmSignUpInput(
                    clientId: clientId,
                    confirmationCode: code,
                    username: userName
                )
            )

            print("=====> \(userName) has been confirmed.")
            return true
        } catch {
            print("=====> \(userName)'s code was entered incorrectly.")
            return false
        }
    }

    /// Begin an authentication session.
    ///
```

```
    /// - Parameters:
    ///   - cipClient: The `CongitoIdentityProviderClient` to use.
    ///   - clientId: The app client ID to use.
    ///   - userName: The username to check.
    ///   - password: The user's password.
    ///   - userPoolId: The user pool to use.
    ///
    /// - Returns: The session token associated with this authentication
    ///   session.
    func initiateAuth(cipClient: CognitoIdentityProviderClient, clientId: String,
                      userName: String, password: String,
                      userPoolId: String) async -> String? {
        var authParams: [String: String] = [:]

        authParams["USERNAME"] = userName
        authParams["PASSWORD"] = password

        do {
            let output = try await cipClient.adminInitiateAuth(
                input: AdminInitiateAuthInput(
                    authFlow:
CognitoIdentityProviderClientTypes.AuthFlowType.adminUserPasswordAuth,
                    authParameters: authParams,
                    clientId: clientId,
                    userPoolId: userPoolId
                )
            )

            guard let challengeName = output.challengeName else {
                print("*** Invalid response from the auth service.")
                return nil
            }

            print("=====> Response challenge is \(challengeName)")

            return output.session
        } catch _ as UserNotFoundException {
            print("*** The specified username, \(userName), doesn't exist.")
            return nil
        } catch _ as UserNotConfirmedException {
            print("*** The user \(userName) has not been confirmed.")
            return nil
        } catch {
            print("*** An unexpected error occurred.")
```

```
            return nil
        }
    }

    /// Request and display an MFA secret token that the user should enter
    /// into their authenticator to set it up for the user account.
    ///
    /// - Parameters:
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - authSession: The authentication session to request an MFA secret
    ///     for.
    ///
    /// - Returns: A string containing the MFA secret token that should be
    ///   entered into the authenticator software.
    func getSecretForAppMFA(cipClient: CognitoIdentityProviderClient,
authSession: String?) async -> String? {
        do {
            let output = try await cipClient.associateSoftwareToken(
                input: AssociateSoftwareTokenInput(
                    session: authSession
                )
            )

            guard let secretCode = output.secretCode else {
                print("*** Unable to get the secret code")
                return nil
            }

            print("=====> Enter this token into Google Authenticator:
\(secretCode)")
            return output.session
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return nil
        } catch {
            print("*** An unexpected error occurred getting the secret for the
app's MFA.")
            return nil
        }
    }

    /// Confirm that the user's TOTP authenticator is configured correctly by
    /// sending a code to it to check that it matches successfully.
    ///
```

```
    /// - Parameters:
    ///   - cipClient: The `CongnitoIdentityProviderClient` to use.
    ///   - session: An authentication session previously returned by an
    ///     `associateSoftwareToken()` call.
    ///   - mfaCode: The 6-digit code currently displayed by the user's
    ///     authenticator, as provided by the user.
    func verifyTOTP(cipClient: CognitoIdentityProviderClient, session: String?,
mfaCode: String?) async {
        do {
            let output = try await cipClient.verifySoftwareToken(
                input: VerifySoftwareTokenInput(
                    session: session,
                    userCode: mfaCode
                )
            )

            guard let tokenStatus = output.status else {
                print("*** Unable to get the token's status.")
                return
            }
            print("=====> The token's status is: \(tokenStatus)")
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return
        } catch _ as CodeMismatchException {
            print("*** The specified MFA code doesn't match the expected value.")
            return
        } catch _ as UserNotFoundException {
            print("*** The specified username doesn't exist.")
            return
        } catch _ as UserNotConfirmedException {
            print("*** The user has not been confirmed.")
            return
        } catch {
            print("*** Error verifying the MFA token!")
            return
        }
    }

    /// Respond to the authentication challenge received from Cognito after
    /// initiating an authentication session. This involves sending a current
    /// MFA code to the service.
    ///
    /// - Parameters:
```

```
    ///   - cipClient: The `CognitoIdentityProviderClient` to use.
    ///   - userName: The user's username.
    ///   - clientId: The app client ID.
    ///   - userPoolId: The user pool to sign into.
    ///   - mfaCode: The 6-digit MFA code currently displayed by the user's
    ///     authenticator.
    ///   - session: The authentication session to continue processing.
    func adminRespondToAuthChallenge(cipClient: CognitoIdentityProviderClient,
userName: String,
                                     clientId: String, userPoolId: String,
mfaCode: String,
                                     session: String) async {
        print("=====> SOFTWARE_TOKEN_MFA challenge is generated...")

        var challengeResponsesOb: [String: String] = [:]
        challengeResponsesOb["USERNAME"] = userName
        challengeResponsesOb["SOFTWARE_TOKEN_MFA_CODE"] = mfaCode

        do {
            let output = try await cipClient.adminRespondToAuthChallenge(
                input: AdminRespondToAuthChallengeInput(
                    challengeName:
CognitoIdentityProviderClientTypes.ChallengeNameType.softwareTokenMfa,
                    challengeResponses: challengeResponsesOb,
                    clientId: clientId,
                    session: session,
                    userPoolId: userPoolId
                )
            )

            guard let authenticationResult = output.authenticationResult else {
                print("*** Unable to get authentication result.")
                return
            }

            print("=====> Authentication result (JWTs are redacted):")
            print(authenticationResult)
        } catch _ as SoftwareTokenMFANotFoundException {
            print("*** The specified user pool isn't configured for MFA.")
            return
        } catch _ as CodeMismatchException {
            print("*** The specified MFA code doesn't match the expected value.")
            return
        } catch _ as UserNotFoundException {
```

```
            print("*** The specified username, \(userName), doesn't exist.")
            return
        } catch _ as UserNotConfirmedException {
            print("*** The user \(userName) has not been confirmed.")
            return
        } catch let error as NotAuthorizedException {
            print("*** Unauthorized access. Reason: \(error.properties.message ??
"<unknown>")")
        } catch {
            print("*** Error responding to the MFA challenge.")
            return
        }
    }

    /// Called by ``main()`` to run the bulk of the example.
    func runAsync() async throws {
        let config = try await
CognitoIdentityProviderClient.CognitoIdentityProviderClientConfig(region:
region)
        let cipClient = CognitoIdentityProviderClient(config: config)

        print("""
              This example collects information about a user, then creates that
user in the
              specified user pool. Then, it enables Multi-Factor Authentication
(MFA) for that
              user by associating an authenticator application (such as Google
Authenticator
              or a password manager that supports TOTP). Then, the user uses a
code from their
              authenticator application to sign in.

              """)

        let userName = stringRequest("Please enter a new username: ")
        let password = stringRequest("Enter a password: ")
        let email = stringRequest("Enter your email address: ", minLength: 5)

        // Submit the sign-up request to AWS.

        print("==> Signing up user \(userName)...")
        if await signUp(cipClient: cipClient, clientId: clientId,
                        userName: userName, password: password,
                        email: email) == false {
```

```
            return
        }

        // Check the user's status. This time, it should come back "unconfirmed".

        print("==> Getting the status of user \(userName) from the user pool
(should be 'unconfirmed')...")
        if await adminGetUser(cipClient: cipClient, userName: userName,
userPoolId: poolId) == false {
            return
        }

        // Ask the user if they want a replacement code sent, such as if the
        // code hasn't arrived yet. If the user responds with a "yes," send a
        // new code.

        if yesNoRequest("==> A confirmation code was sent to \(userName). Would
you like to send a new code (Y/N)? ") {
            print("==> Sending a new confirmation code...")
            if await resendConfirmationCode(cipClient: cipClient, clientId:
clientId, userName: userName) == false {
                return
            }
        }

        // Ask the user to enter the confirmation code, then send it to Amazon
        // Cognito to verify it.

        let code = stringRequest("==> Enter the confirmation code sent to
\(userName): ")
        if await confirmSignUp(cipClient: cipClient, clientId: clientId,
userName: userName, code: code) == false {
            // The code didn't match. Your application may wish to offer to
            // re-send the confirmation code here and try again.
            return
        }

        // Check the user's status again. This time it should come back
        // "confirmed".

        print("==> Rechecking status of user \(userName) in the user pool (should
be 'confirmed')...")
        if await adminGetUser(cipClient: cipClient, userName: userName,
userPoolId: poolId) == false {
```

```
            return
        }
        // Check the challenge mode. Here, it should be "mfaSetup", indicating
        // that the user needs to add MFA before using it. This returns a
        // session that can be used to register MFA, or nil if an error occurs.

        let authSession = await initiateAuth(cipClient: cipClient, clientId:
clientId,
                                             userName: userName, password:
password,
                                             userPoolId: poolId)
        if authSession == nil {
            return
        }

        // Ask Cognito for an MFA secret token that the user should enter into
        // their authenticator software (such as Google Authenticator) or
        // password manager to configure it for this user account. This
        // returns a new session that should be used for the new stage of the
        // authentication process.

        let newSession = await getSecretForAppMFA(cipClient: cipClient,
authSession: authSession)
        if newSession == nil {
            return
        }

        // Ask the user to enter the current 6-digit code displayed by their
        // authenticator. Then verify that it matches the value expected for
        // the session.

        let mfaCode1 = stringRequest("==> Enter the 6-digit code displayed in
your authenticator: ",
                                     minLength: 6)
        await verifyTOTP(cipClient: cipClient, session: newSession, mfaCode:
mfaCode1)

        // Ask the user to authenticate now that the authenticator has been
        // configured. This creates a new session using the user's username
        // and password as already entered.

        print("\nNow starting the sign-in process for user \(userName)...\n")
```

```
        let session2 = await initiateAuth(cipClient: cipClient, clientId:
 clientId,
                                          userName: userName, password: password,
 userPoolId: poolId)
        guard let session2 else {
            return
        }

        // Now that we have a new auth session, `session2`, ask the user for a
        // new 6-digit code from their authenticator, and send it to the auth
        // session.

        let mfaCode2 = stringRequest("==> Wait for your authenticator to show a
 new 6-digit code, then enter it: ",
                                     minLength: 6)
        await adminRespondToAuthChallenge(cipClient: cipClient, userName:
 userName,
                                          clientId: clientId, userPoolId: poolId,
                                          mfaCode: mfaCode2, session: session2)
    }
}

/// The program's asynchronous entry point.
@main
struct Main {
    static func main() async {
        let args = Array(CommandLine.arguments.dropFirst())

        do {
            let command = try ExampleCommand.parse(args)
            try await command.runAsync()
        } catch {
            ExampleCommand.exit(withError: error)
        }
    }
}
```

- For API details, see the following topics in *AWS SDK for Swift API reference*.
  - AdminGetUser
  - AdminInitiateAuth
  - AdminRespondToAuthChallenge

- AssociateSoftwareToken
- ConfirmDevice
- ConfirmSignUp
- InitiateAuth
- ListUsers
- ResendConfirmationCode
- RespondToAuthChallenge
- SignUp
- VerifySoftwareToken

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Use Amazon Cognito identity pools and authentication flows

The following code example shows how to create a web-based demo application that demonstrates identity pools authentication flows.

Python

SDK for Python (Boto3)

Shows web-based demo application that demonstrates Amazon Cognito identity pools authentication flows, allowing users to interactively explore both enhanced and basic authentication flows with various identity providers.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- Amazon Cognito Identity Provider

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Write custom activity data with a Lambda function after Amazon Cognito user authentication using an AWS SDK

The following code example shows how to write custom activity data with a Lambda function after Amazon Cognito user authentication.

- Use administrator functions to add a user to a user pool.
- Configure a user pool to call a Lambda function for the `PostAuthentication` trigger.
- Sign the new user in to Amazon Cognito.
- The Lambda function writes custom information to CloudWatch Logs and to an DynamoDB table.
- Get and display custom data from the DynamoDB table, then clean up resources.

Go

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

Run an interactive scenario at a command prompt.

```
import (
 "context"
 "errors"
 "log"
 "strings"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)
```

```
// ActivityLog separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ActivityLog struct {
 helper       IScenarioHelper
 questioner   demotools.IQuestioner
 resources    Resources
 cognitoActor *actions.CognitoActions
}

// NewActivityLog constructs a new activity log runner.
func NewActivityLog(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) ActivityLog {
 scenario := ActivityLog{
  helper:       helper,
  questioner:   questioner,
  resources:    Resources{},
  cognitoActor: &actions.CognitoActions{CognitoClient:
 cognitoidentityprovider.NewFromConfig(sdkConfig)},
 }
 scenario.resources.init(scenario.cognitoActor, questioner)
 return scenario
}

// AddUserToPool selects a user from the known users table and uses administrator
 credentials to add the user to the user pool.
func (runner *ActivityLog) AddUserToPool(ctx context.Context, userPoolId string,
 tableName string) (string, string) {
 log.Println("To facilitate this example, let's add a user to the user pool using
 administrator privileges.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 user := users.Users[0]
 log.Printf("Adding known user %v to the user pool.\n", user.UserName)
 err = runner.cognitoActor.AdminCreateUser(ctx, userPoolId, user.UserName,
 user.UserEmail)
 if err != nil {
  panic(err)
 }
 pwSet := false
 password := runner.questioner.AskPassword("\nEnter a password that has at least
 eight characters, uppercase, lowercase, numbers and symbols.\n"+
```

```
  "(the password will not display as you type):", 8)
 for !pwSet {
  log.Printf("\nSetting password for user '%v'.\n", user.UserName)
  err = runner.cognitoActor.AdminSetUserPassword(ctx, userPoolId, user.UserName,
 password)
  if err != nil {
   var invalidPassword *types.InvalidPasswordException
   if errors.As(err, &invalidPassword) {
    password = runner.questioner.AskPassword("\nEnter another password:", 8)
   } else {
    panic(err)
   }
  } else {
   pwSet = true
  }
 }

 log.Println(strings.Repeat("-", 88))

 return user.UserName, password
}

// AddActivityLogTrigger adds a Lambda handler as an invocation target for the
 PostAuthentication trigger.
func (runner *ActivityLog) AddActivityLogTrigger(ctx context.Context, userPoolId
 string, activityLogArn string) {
 log.Println("Let's add a Lambda function to handle the PostAuthentication
 trigger from Cognito.\n" +
  "This trigger happens after a user is authenticated, and lets your function
 take action, such as logging\n" +
  "the outcome.")
 err := runner.cognitoActor.UpdateTriggers(
  ctx, userPoolId,
  actions.TriggerInfo{Trigger: actions.PostAuthentication, HandlerArn:
 aws.String(activityLogArn)})
 if err != nil {
  panic(err)
 }
 runner.resources.triggers = append(runner.resources.triggers,
 actions.PostAuthentication)
 log.Printf("Lambda function %v added to user pool %v to handle
 PostAuthentication Cognito trigger.\n",
  activityLogArn, userPoolId)
```

```
 log.Println(strings.Repeat("-", 88))
}

// SignInUser signs in as the specified user.
func (runner *ActivityLog) SignInUser(ctx context.Context, clientId string,
 userName string, password string) {
 log.Printf("Now we'll sign in user %v and check the results in the logs and the
 DynamoDB table.", userName)
 runner.questioner.Ask("Press Enter when you're ready.")
 authResult, err := runner.cognitoActor.SignIn(ctx, clientId, userName, password)
 if err != nil {
  panic(err)
 }
 log.Println("Sign in successful.",
  "The PostAuthentication Lambda handler writes custom information to CloudWatch
 Logs.")

 runner.resources.userAccessTokens = append(runner.resources.userAccessTokens,
 *authResult.AccessToken)
}

// GetKnownUserLastLogin gets the login info for a user from the Amazon DynamoDB
 table and displays it.
func (runner *ActivityLog) GetKnownUserLastLogin(ctx context.Context, tableName
 string, userName string) {
 log.Println("The PostAuthentication handler also writes login data to the
 DynamoDB table.")
 runner.questioner.Ask("Press Enter when you're ready to continue.")
 users, err := runner.helper.GetKnownUsers(ctx, tableName)
 if err != nil {
  panic(err)
 }
 for _, user := range users.Users {
  if user.UserName == userName {
   log.Println("The last login info for the user in the known users table is:")
   log.Printf("\t%+v", *user.LastLogin)
  }
 }
 log.Println(strings.Repeat("-", 88))
}

// Run runs the scenario.
func (runner *ActivityLog) Run(ctx context.Context, stackName string) {
 defer func() {
```

```
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   runner.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome\n")

 log.Println(strings.Repeat("-", 88))

 stackOutputs, err := runner.helper.GetStackOutputs(ctx, stackName)
 if err != nil {
  panic(err)
 }
 runner.resources.userPoolId = stackOutputs["UserPoolId"]
 runner.helper.PopulateUserTable(ctx, stackOutputs["TableName"])
 userName, password := runner.AddUserToPool(ctx, stackOutputs["UserPoolId"],
 stackOutputs["TableName"])

 runner.AddActivityLogTrigger(ctx, stackOutputs["UserPoolId"],
 stackOutputs["ActivityLogFunctionArn"])
 runner.SignInUser(ctx, stackOutputs["UserPoolClientId"], userName, password)
 runner.helper.ListRecentLogEvents(ctx, stackOutputs["ActivityLogFunction"])
 runner.GetKnownUserLastLogin(ctx, stackOutputs["TableName"], userName)

 runner.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Handle the `PostAuthentication` trigger with a Lambda function.

```
import (
 "context"
 "fmt"
 "log"
 "os"
```

```
 "time"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 dynamodbtypes "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

const TABLE_NAME = "TABLE_NAME"

// LoginInfo defines structured login data that can be marshalled to a DynamoDB
 format.
type LoginInfo struct {
 UserPoolId string `dynamodbav:"UserPoolId"`
 ClientId   string `dynamodbav:"ClientId"`
 Time       string `dynamodbav:"Time"`
}

// UserInfo defines structured user data that can be marshalled to a DynamoDB
 format.
type UserInfo struct {
 UserName  string    `dynamodbav:"UserName"`
 UserEmail string    `dynamodbav:"UserEmail"`
 LastLogin LoginInfo `dynamodbav:"LastLogin"`
}

// GetKey marshals the user email value to a DynamoDB key format.
func (user UserInfo) GetKey() map[string]dynamodbtypes.AttributeValue {
 userEmail, err := attributevalue.Marshal(user.UserEmail)
 if err != nil {
  panic(err)
 }
 return map[string]dynamodbtypes.AttributeValue{"UserEmail": userEmail}
}

type handler struct {
 dynamoClient *dynamodb.Client
}

// HandleRequest handles the PostAuthentication event by writing custom data to
 the logs and
```

```
// to an Amazon DynamoDB table.
func (h *handler) HandleRequest(ctx context.Context,
 event events.CognitoEventUserPoolsPostAuthentication)
 (events.CognitoEventUserPoolsPostAuthentication, error) {
 log.Printf("Received post authentication trigger from %v for user '%v'",
 event.TriggerSource, event.UserName)
 tableName := os.Getenv(TABLE_NAME)
 user := UserInfo{
  UserName:  event.UserName,
  UserEmail: event.Request.UserAttributes["email"],
  LastLogin: LoginInfo{
   UserPoolId: event.UserPoolID,
   ClientId:   event.CallerContext.ClientID,
   Time:       time.Now().Format(time.UnixDate),
  },
 }
 // Write to CloudWatch Logs.
 fmt.Printf("%#v", user)

 // Also write to an external system. This examples uses DynamoDB to demonstrate.
 userMap, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshal to DynamoDB map. Here's why: %v\n", err)
 } else if len(userMap) == 0 {
  log.Printf("User info marshaled to an empty map.")
 } else {
  _, err := h.dynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
   Item:      userMap,
   TableName: aws.String(tableName),
  })
  if err != nil {
   log.Printf("Couldn't write to DynamoDB. Here's why: %v\n", err)
  } else {
   log.Printf("Wrote user info to DynamoDB table %v.\n", tableName)
  }
 }

 return event, nil
}

func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
```

```
  log.Panicln(err)
 }
 h := handler{
  dynamoClient: dynamodb.NewFromConfig(sdkConfig),
 }
 lambda.Start(h.HandleRequest)
}
```

Create a struct that performs common tasks.

```
import (
 "context"
 "log"
 "strings"
 "time"
 "user_pools_and_lambda_triggers/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// IScenarioHelper defines common functions used by the workflows in this
 example.
type IScenarioHelper interface {
 Pause(secs int)
 GetStackOutputs(ctx context.Context, stackName string) (actions.StackOutputs,
 error)
 PopulateUserTable(ctx context.Context, tableName string)
 GetKnownUsers(ctx context.Context, tableName string) (actions.UserList, error)
 AddKnownUser(ctx context.Context, tableName string, user actions.User)
 ListRecentLogEvents(ctx context.Context, functionName string)
}

// ScenarioHelper contains AWS wrapper structs used by the workflows in this
 example.
type ScenarioHelper struct {
 questioner  demotools.IQuestioner
```

```
 dynamoActor *actions.DynamoActions
 cfnActor    *actions.CloudFormationActions
 cwlActor    *actions.CloudWatchLogsActions
 isTestRun   bool
}

// NewScenarioHelper constructs a new scenario helper.
func NewScenarioHelper(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ScenarioHelper {
 scenario := ScenarioHelper{
  questioner:  questioner,
  dynamoActor: &actions.DynamoActions{DynamoClient:
 dynamodb.NewFromConfig(sdkConfig)},
  cfnActor:    &actions.CloudFormationActions{CfnClient:
 cloudformation.NewFromConfig(sdkConfig)},
  cwlActor:    &actions.CloudWatchLogsActions{CwlClient:
 cloudwatchlogs.NewFromConfig(sdkConfig)},
 }
 return scenario
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 if !helper.isTestRun {
  time.Sleep(time.Duration(secs) * time.Second)
 }
}

// GetStackOutputs gets the outputs from the specified CloudFormation stack in a
 structured format.
func (helper ScenarioHelper) GetStackOutputs(ctx context.Context, stackName
 string) (actions.StackOutputs, error) {
 return helper.cfnActor.GetOutputs(ctx, stackName), nil
}

// PopulateUserTable fills the known user table with example data.
func (helper ScenarioHelper) PopulateUserTable(ctx context.Context, tableName
 string) {
 log.Printf("First, let's add some users to the DynamoDB %v table we'll use for
 this example.\n", tableName)
 err := helper.dynamoActor.PopulateTable(ctx, tableName)
 if err != nil {
  panic(err)
 }
```

```
}

// GetKnownUsers gets the users from the known users table in a structured
 format.
func (helper ScenarioHelper) GetKnownUsers(ctx context.Context, tableName string)
 (actions.UserList, error) {
 knownUsers, err := helper.dynamoActor.Scan(ctx, tableName)
 if err != nil {
  log.Printf("Couldn't get known users from table %v. Here's why: %v\n",
 tableName, err)
 }
 return knownUsers, err
}

// AddKnownUser adds a user to the known users table.
func (helper ScenarioHelper) AddKnownUser(ctx context.Context, tableName string,
 user actions.User) {
 log.Printf("Adding user '%v' with email '%v' to the DynamoDB known users
 table...\n",
  user.UserName, user.UserEmail)
 err := helper.dynamoActor.AddUser(ctx, tableName, user)
 if err != nil {
  panic(err)
 }
}

// ListRecentLogEvents gets the most recent log stream and events for the
 specified Lambda function and displays them.
func (helper ScenarioHelper) ListRecentLogEvents(ctx context.Context,
 functionName string) {
 log.Println("Waiting a few seconds to let Lambda write to CloudWatch Logs...")
 helper.Pause(10)
 log.Println("Okay, let's check the logs to find what's happened recently with
 your Lambda function.")
 logStream, err := helper.cwlActor.GetLatestLogStream(ctx, functionName)
 if err != nil {
  panic(err)
 }
 log.Printf("Getting some recent events from log stream %v\n",
 *logStream.LogStreamName)
 events, err := helper.cwlActor.GetLogEvents(ctx, functionName,
 *logStream.LogStreamName, 10)
 if err != nil {
  panic(err)
```

```
 }
 for _, event := range events {
  log.Printf("\t%v", *event.Message)
 }
 log.Println(strings.Repeat("-", 88))
}
```

Create a struct that wraps Amazon Cognito actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider"
 "github.com/aws/aws-sdk-go-v2/service/cognitoidentityprovider/types"
)

type CognitoActions struct {
 CognitoClient *cognitoidentityprovider.Client
}



// Trigger and TriggerInfo define typed data for updating an Amazon Cognito
 trigger.
type Trigger int

const (
 PreSignUp Trigger = iota
 UserMigration
 PostAuthentication
)

type TriggerInfo struct {
 Trigger    Trigger
 HandlerArn *string
}
```

```
// UpdateTriggers adds or removes Lambda triggers for a user pool. When a trigger
 is specified with a `nil` value,
// it is removed from the user pool.
func (actor CognitoActions) UpdateTriggers(ctx context.Context, userPoolId
 string, triggers ...TriggerInfo) error {
 output, err := actor.CognitoClient.DescribeUserPool(ctx,
 &cognitoidentityprovider.DescribeUserPoolInput{
  UserPoolId: aws.String(userPoolId),
 })
 if err != nil {
  log.Printf("Couldn't get info about user pool %v. Here's why: %v\n",
 userPoolId, err)
  return err
 }
 lambdaConfig := output.UserPool.LambdaConfig
 for _, trigger := range triggers {
  switch trigger.Trigger {
  case PreSignUp:
   lambdaConfig.PreSignUp = trigger.HandlerArn
  case UserMigration:
   lambdaConfig.UserMigration = trigger.HandlerArn
  case PostAuthentication:
   lambdaConfig.PostAuthentication = trigger.HandlerArn
  }
 }
 _, err = actor.CognitoClient.UpdateUserPool(ctx,
 &cognitoidentityprovider.UpdateUserPoolInput{
  UserPoolId:   aws.String(userPoolId),
  LambdaConfig: lambdaConfig,
 })
 if err != nil {
  log.Printf("Couldn't update user pool %v. Here's why: %v\n", userPoolId, err)
 }
 return err
}



// SignUp signs up a user with Amazon Cognito.
func (actor CognitoActions) SignUp(ctx context.Context, clientId string, userName
 string, password string, userEmail string) (bool, error) {
 confirmed := false
 output, err := actor.CognitoClient.SignUp(ctx,
 &cognitoidentityprovider.SignUpInput{
```

```
  ClientId: aws.String(clientId),
  Password: aws.String(password),
  Username: aws.String(userName),
  UserAttributes: []types.AttributeType{
   {Name: aws.String("email"), Value: aws.String(userEmail)},
  },
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't sign up user %v. Here's why: %v\n", userName, err)
  }
 } else {
  confirmed = output.UserConfirmed
 }
 return confirmed, err
}



// SignIn signs in a user to Amazon Cognito using a username and password
 authentication flow.
func (actor CognitoActions) SignIn(ctx context.Context, clientId string, userName
 string, password string) (*types.AuthenticationResultType, error) {
 var authResult *types.AuthenticationResultType
 output, err := actor.CognitoClient.InitiateAuth(ctx,
 &cognitoidentityprovider.InitiateAuthInput{
  AuthFlow:       "USER_PASSWORD_AUTH",
  ClientId:       aws.String(clientId),
  AuthParameters: map[string]string{"USERNAME": userName, "PASSWORD": password},
 })
 if err != nil {
  var resetRequired *types.PasswordResetRequiredException
  if errors.As(err, &resetRequired) {
   log.Println(*resetRequired.Message)
  } else {
   log.Printf("Couldn't sign in user %v. Here's why: %v\n", userName, err)
  }
 } else {
  authResult = output.AuthenticationResult
 }
 return authResult, err
```

```
}



// ForgotPassword starts a password recovery flow for a user. This flow typically
 sends a confirmation code
// to the user's configured notification destination, such as email.
func (actor CognitoActions) ForgotPassword(ctx context.Context, clientId string,
 userName string) (*types.CodeDeliveryDetailsType, error) {
 output, err := actor.CognitoClient.ForgotPassword(ctx,
 &cognitoidentityprovider.ForgotPasswordInput{
  ClientId: aws.String(clientId),
  Username: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't start password reset for user '%v'. Here;s why: %v\n",
 userName, err)
 }
 return output.CodeDeliveryDetails, err
}



// ConfirmForgotPassword confirms a user with a confirmation code and a new
 password.
func (actor CognitoActions) ConfirmForgotPassword(ctx context.Context, clientId
 string, code string, userName string, password string) error {
 _, err := actor.CognitoClient.ConfirmForgotPassword(ctx,
 &cognitoidentityprovider.ConfirmForgotPasswordInput{
  ClientId:         aws.String(clientId),
  ConfirmationCode: aws.String(code),
  Password:         aws.String(password),
  Username:         aws.String(userName),
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't confirm user %v. Here's why: %v", userName, err)
  }
 }
 return err
}
```

```
// DeleteUser removes a user from the user pool.
func (actor CognitoActions) DeleteUser(ctx context.Context, userAccessToken
 string) error {
 _, err := actor.CognitoClient.DeleteUser(ctx,
 &cognitoidentityprovider.DeleteUserInput{
  AccessToken: aws.String(userAccessToken),
 })
 if err != nil {
  log.Printf("Couldn't delete user. Here's why: %v\n", err)
 }
 return err
}



// AdminCreateUser uses administrator credentials to add a user to a user pool.
 This method leaves the user
// in a state that requires they enter a new password next time they sign in.
func (actor CognitoActions) AdminCreateUser(ctx context.Context, userPoolId
 string, userName string, userEmail string) error {
 _, err := actor.CognitoClient.AdminCreateUser(ctx,
 &cognitoidentityprovider.AdminCreateUserInput{
  UserPoolId:     aws.String(userPoolId),
  Username:       aws.String(userName),
  MessageAction:  types.MessageActionTypeSuppress,
  UserAttributes: []types.AttributeType{{Name: aws.String("email"), Value:
 aws.String(userEmail)}},
 })
 if err != nil {
  var userExists *types.UsernameExistsException
  if errors.As(err, &userExists) {
   log.Printf("User %v already exists in the user pool.", userName)
   err = nil
  } else {
   log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
  }
 }
 return err
}
```

```
// AdminSetUserPassword uses administrator credentials to set a password for a
 user without requiring a
// temporary password.
func (actor CognitoActions) AdminSetUserPassword(ctx context.Context, userPoolId
 string, userName string, password string) error {
 _, err := actor.CognitoClient.AdminSetUserPassword(ctx,
 &cognitoidentityprovider.AdminSetUserPasswordInput{
  Password:   aws.String(password),
  UserPoolId: aws.String(userPoolId),
  Username:   aws.String(userName),
  Permanent:  true,
 })
 if err != nil {
  var invalidPassword *types.InvalidPasswordException
  if errors.As(err, &invalidPassword) {
   log.Println(*invalidPassword.Message)
  } else {
   log.Printf("Couldn't set password for user %v. Here's why: %v\n", userName,
 err)
  }
 }
 return err
}
```

Create a struct that wraps DynamoDB actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// DynamoActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
```

```
// used in the examples.
type DynamoActions struct {
 DynamoClient *dynamodb.Client
}

// User defines structured user data.
type User struct {
 UserName  string
 UserEmail string
 LastLogin *LoginInfo `dynamodbav:",omitempty"`
}

// LoginInfo defines structured custom login data.
type LoginInfo struct {
 UserPoolId string
 ClientId   string
 Time       string
}

// UserList defines a list of users.
type UserList struct {
 Users []User
}

// UserNameList returns the usernames contained in a UserList as a list of
 strings.
func (users *UserList) UserNameList() []string {
 names := make([]string, len(users.Users))
 for i := 0; i < len(users.Users); i++ {
  names[i] = users.Users[i].UserName
 }
 return names
}

// PopulateTable adds a set of test users to the table.
func (actor DynamoActions) PopulateTable(ctx context.Context, tableName string)
 error {
 var err error
 var item map[string]types.AttributeValue
 var writeReqs []types.WriteRequest
 for i := 1; i < 4; i++ {
  item, err = attributevalue.MarshalMap(User{UserName: fmt.Sprintf("test_user_
%v", i), UserEmail: fmt.Sprintf("test_email_%v@example.com", i)})
  if err != nil {
```

```
    log.Printf("Couldn't marshall user into DynamoDB format. Here's why: %v\n",
 err)
    return err
   }
   writeReqs = append(writeReqs, types.WriteRequest{PutRequest:
 &types.PutRequest{Item: item}})
 }
 _, err = actor.DynamoClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
  RequestItems: map[string][]types.WriteRequest{tableName: writeReqs},
 })
 if err != nil {
  log.Printf("Couldn't populate table %v with users. Here's why: %v\n",
 tableName, err)
 }
 return err
}

// Scan scans the table for all items.
func (actor DynamoActions) Scan(ctx context.Context, tableName string) (UserList,
 error) {
 var userList UserList
 output, err := actor.DynamoClient.Scan(ctx, &dynamodb.ScanInput{
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't scan table %v for items. Here's why: %v\n", tableName,
 err)
 } else {
  err = attributevalue.UnmarshalListOfMaps(output.Items, &userList.Users)
  if err != nil {
   log.Printf("Couldn't unmarshal items into users. Here's why: %v\n", err)
  }
 }
 return userList, err
}

// AddUser adds a user item to a table.
func (actor DynamoActions) AddUser(ctx context.Context, tableName string, user
 User) error {
 userItem, err := attributevalue.MarshalMap(user)
 if err != nil {
  log.Printf("Couldn't marshall user to item. Here's why: %v\n", err)
 }
 _, err = actor.DynamoClient.PutItem(ctx, &dynamodb.PutItemInput{
```

```
  Item:      userItem,
  TableName: aws.String(tableName),
 })
 if err != nil {
  log.Printf("Couldn't put item in table %v. Here's why: %v", tableName, err)
 }
 return err
}
```

Create a struct that wraps CloudWatch Logs actions.

```
import (
 "context"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)

type CloudWatchLogsActions struct {
 CwlClient *cloudwatchlogs.Client
}

// GetLatestLogStream gets the most recent log stream for a Lambda function.
func (actor CloudWatchLogsActions) GetLatestLogStream(ctx context.Context,
 functionName string) (types.LogStream, error) {
 var logStream types.LogStream
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.DescribeLogStreams(ctx,
 &cloudwatchlogs.DescribeLogStreamsInput{
  Descending:   aws.Bool(true),
  Limit:        aws.Int32(1),
  LogGroupName: aws.String(logGroupName),
  OrderBy:      types.OrderByLastEventTime,
 })
 if err != nil {
  log.Printf("Couldn't get log streams for log group %v. Here's why: %v\n",
 logGroupName, err)
```

```
 } else {
  logStream = output.LogStreams[0]
 }
 return logStream, err
}

// GetLogEvents gets the most recent eventCount events from the specified log
 stream.
func (actor CloudWatchLogsActions) GetLogEvents(ctx context.Context, functionName
 string, logStreamName string, eventCount int32) (
 []types.OutputLogEvent, error) {
 var events []types.OutputLogEvent
 logGroupName := fmt.Sprintf("/aws/lambda/%s", functionName)
 output, err := actor.CwlClient.GetLogEvents(ctx,
 &cloudwatchlogs.GetLogEventsInput{
  LogStreamName: aws.String(logStreamName),
  Limit:         aws.Int32(eventCount),
  LogGroupName:  aws.String(logGroupName),
 })
 if err != nil {
  log.Printf("Couldn't get log event for log stream %v. Here's why: %v\n",
 logStreamName, err)
 } else {
  events = output.Events
 }
 return events, err
}
```

Create a struct that wraps CloudFormation actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/cloudformation"
)

// StackOutputs defines a map of outputs from a specific stack.
type StackOutputs map[string]string
```

```
type CloudFormationActions struct {
 CfnClient *cloudformation.Client
}

// GetOutputs gets the outputs from a CloudFormation stack and puts them into a
 structured format.
func (actor CloudFormationActions) GetOutputs(ctx context.Context, stackName
 string) StackOutputs {
 output, err := actor.CfnClient.DescribeStacks(ctx,
 &cloudformation.DescribeStacksInput{
  StackName: aws.String(stackName),
 })
 if err != nil || len(output.Stacks) == 0 {
  log.Panicf("Couldn't find a CloudFormation stack named %v. Here's why: %v\n",
 stackName, err)
 }
 stackOutputs := StackOutputs{}
 for _, out := range output.Stacks[0].Outputs {
  stackOutputs[*out.OutputKey] = *out.OutputValue
 }
 return stackOutputs
}
```

Clean up resources.

```
import (
 "context"
 "log"
 "user_pools_and_lambda_triggers/actions"

 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 userPoolId       string
 userAccessTokens []string
 triggers         []actions.Trigger
```

```
 cognitoActor *actions.CognitoActions
 questioner   demotools.IQuestioner
}

func (resources *Resources) init(cognitoActor *actions.CognitoActions, questioner
 demotools.IQuestioner) {
 resources.userAccessTokens = []string{}
 resources.triggers = []actions.Trigger{}
 resources.cognitoActor = cognitoActor
 resources.questioner = questioner
}

// Cleanup deletes all AWS resources created during an example.
func (resources *Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong during cleanup.\n%v\n", r)
   log.Println("Use the AWS Management Console to remove any remaining resources
 \n" +
    "that were created for this scenario.")
  }
 }()

 wantDelete := resources.questioner.AskBool("Do you want to remove all of the AWS
 resources that were created "+
  "during this demo (y/n)?", "y")
 if wantDelete {
  for _, accessToken := range resources.userAccessTokens {
   err := resources.cognitoActor.DeleteUser(ctx, accessToken)
   if err != nil {
    log.Println("Couldn't delete user during cleanup.")
    panic(err)
   }
   log.Println("Deleted user.")
  }
  triggerList := make([]actions.TriggerInfo, len(resources.triggers))
  for i := 0; i < len(resources.triggers); i++ {
   triggerList[i] = actions.TriggerInfo{Trigger: resources.triggers[i],
 HandlerArn: nil}
  }
  err := resources.cognitoActor.UpdateTriggers(ctx, resources.userPoolId,
 triggerList...)
  if err != nil {
```

```
    log.Println("Couldn't update Cognito triggers during cleanup.")
    panic(err)
   }
   log.Println("Removed Cognito triggers from user pool.")
  } else {
   log.Println("Be sure to remove resources when you're done with them to avoid
  unexpected charges!")
  }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - AdminCreateUser
  - AdminSetUserPassword
  - DeleteUser
  - InitiateAuth
  - UpdateUserPool

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Code examples for Amazon Cognito Sync using AWS SDKs

The following code examples show how to use Amazon Cognito Sync with an AWS software development kit (SDK).

Actions are code excerpts from larger programs and must be run in context. While actions show you how to call individual service functions, you can see actions in context in their related scenarios.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Amazon Cognito Sync

- Basic examples for Amazon Cognito Sync using AWS SDKs
  - Actions for Amazon Cognito Sync using AWS SDKs

- Use ListIdentityPoolUsage with an AWS SDK

Basic examples for Amazon Cognito Sync using AWS SDKs

The following code examples show how to use the basics of Amazon Cognito Sync with AWS SDKs.

Examples

- Actions for Amazon Cognito Sync using AWS SDKs
  - Use ListIdentityPoolUsage with an AWS SDK

Actions for Amazon Cognito Sync using AWS SDKs

The following code examples demonstrate how to perform individual Amazon Cognito Sync actions with AWS SDKs. Each example includes a link to GitHub, where you can find instructions for setting up and running the code.

The following examples include only the most commonly used actions. For a complete list, see the Amazon Cognito Sync API Reference.

Examples

- Use ListIdentityPoolUsage with an AWS SDK

Use `ListIdentityPoolUsage` with an AWS SDK

The following code example shows how to use `ListIdentityPoolUsage`.

Rust

SDK for Rust

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the AWS Code Examples Repository.

```
async fn show_pools(client: &Client) -> Result<(), Error> {
```

```
    let response = client
        .list_identity_pool_usage()
        .max_results(10)
        .send()
        .await?;

    let pools = response.identity_pool_usages();
    println!("Identity pools:");

    for pool in pools {
        println!(
            "  Identity pool ID:    {}",
            pool.identity_pool_id().unwrap_or_default()
        );
        println!(
            "  Data storage:        {}",
            pool.data_storage().unwrap_or_default()
        );
        println!(
            "  Sync sessions count: {}",
            pool.sync_sessions_count().unwrap_or_default()
        );
        println!(
            "  Last modified:       {}",
            pool.last_modified_date().unwrap().to_chrono_utc()?
        );
        println!();
    }

    println!("Next token: {}", response.next_token().unwrap_or_default());

    Ok(())
}
```

- For API details, see ListIdentityPoolUsage in *AWS SDK for Rust API reference*.

For a complete list of AWS SDK developer guides and code examples, see Using this service with an AWS SDK. This topic also includes information about getting started and details about previous SDK versions.

Other capabilities

Beyond signing users in to your application with a user pool, Amazon Cognito offers additional capabilities for specialized needs. Amazon Cognito identity pools exchange a user's identity for temporary, limited-privilege AWS credentials so that your application can call AWS services directly.

Topics

- Amazon Cognito identity pools
- Getting started with Amazon Cognito identity pools

Amazon Cognito identity pools

An Amazon Cognito identity pool is a directory of federated identities that you can exchange for AWS credentials. Identity pools generate temporary AWS credentials for the users of your app, whether they've signed in or you haven't identified them yet. With AWS Identity and Access Management (IAM) roles and policies, you can choose the level of permission that you want to grant to your users. Users can start out as guests and retrieve assets that you keep in AWS services. Then they can sign in with a third-party identity provider to unlock access to assets that you make available to registered members. The third-party identity provider can be a consumer (social) OAuth 2.0 provider like Apple or Google, a custom SAML or OIDC identity provider, or a custom authentication scheme, also called a *developer provider*, of your own design.

Features of Amazon Cognito identity pools

Sign requests for AWS services

Sign API requests to AWS services like Amazon Simple Storage Service (Amazon S3) and Amazon DynamoDB. Analyze user activity with services like Amazon Pinpoint and Amazon CloudWatch.

Filter requests with resource-based policies

Exercise granular control over user access to your resources. Transform user claims into IAM session tags, and build IAM policies that grant resource access to distinct subsets of your users.

Assign guest access

For your users who haven't signed in yet, configure your identity pool to generate AWS credentials with a narrow scope of access. Authenticate users through a single sign-on provider to elevate their access.

Assign IAM roles based on user characteristics

Assign a single IAM role to all of your authenticated users, or choose the role based on the claims of each user.

Accept a variety of identity providers

Exchange an ID or access token, a user pool token, a SAML assertion, or a social-provider OAuth token for AWS credentials.

Validate your own identities

Perform your own user validation and use your developer AWS credentials to issue credentials for your users.

You might already have an Amazon Cognito user pool that provides authentication and authorization services to your app. You can set up your user pool as an identity provider (IdP) to your identity pool. When you do, your users can authenticate through your user pool IdPs, consolidate their claims into a common OIDC identity token, and exchange that token for AWS credentials. Your user can then present their credentials in a signed request to your AWS services.

You can also present authenticated claims from any of your identity providers directly to your identity pool. Amazon Cognito customizes user claims from SAML, OAuth, and OIDC providers into an AssumeRoleWithWebIdentity API request for short-term credentials.

Amazon Cognito user pools are like OIDC identity providers to your SSO-enabled apps. Identity pools act as an *AWS* identity provider to any app with resource dependencies that work best with IAM authorization.

Amazon Cognito identity pools support the following identity providers:

- Public providers: Setting up Login with Amazon as an identity pools IdP, Setting up Facebook as an identity pools IdP, Setting up Google as an identity pool IdP, Setting up Sign in with Apple as an identity pool IdP, Twitter.
- Amazon Cognito user pools

- Setting up an OIDC provider as an identity pool IdP
- Setting up a SAML provider as an identity pool IdP
- Developer-authenticated identities

For information about Amazon Cognito identity pools Region availability, see AWS Service Region Availability.

For more information about Amazon Cognito identity pools, see the following topics.

Topics

- Identity pools console overview
- Identity pools authentication flow
- IAM roles
- Security best practices for Amazon Cognito identity pools
- Using attributes for access control
- Using role-based access control
- Getting credentials
- Accessing AWS services with temporary credentials
- Identity pools third-party identity providers
- Developer-authenticated identities
- Switching unauthenticated users to authenticated users

Identity pools console overview

Amazon Cognito identity pools provide temporary AWS credentials for users who are guests (unauthenticated) and for users who have been authenticated and received a token. An identity pool is a store of user identifiers linked to your external identity providers.

One way to understand the features and options of identity pools is to create one in the Amazon Cognito console. You can explore the effect of different settings on authentication flows, role-based and attribute-based access control, and guest access. From there, you can proceed to later chapters in this guide and add the appropriate components to your application so that you can implement identity pool authentication.

Topics

- Create an identity pool
- User IAM roles
- Authenticated and unauthenticated identities
- Activate or deactivate guest access
- Change the role associated with an identity type
- Edit identity providers
- Delete an identity pool
- Delete an identity from an identity pool

Create an identity pool

To create a new identity pool in the console

1. Sign in to the Amazon Cognito console and select **Identity pools**.
2. Choose **Create identity pool**.
3. In **Configure identity pool trust**, choose to set up your identity pool for **Authenticated access**, **Guest access**, or both.

    - If you chose **Authenticated access**, select one or more **Identity types** that you want to set as the source of authenticated identities in your identity pool. If you configure a **Custom developer provider**, you can't modify or delete it after you create your identity pool.

4. In **Configure permissions**, choose a default IAM role for authenticated or guest users in your identity pool.

    a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.

    b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an

authenticated user in your specific identity pool. For more information, see Role trust and permissions.

5. In **Connect identity providers**, enter the details of the identity providers (IdPs) that you chose in **Configure identity pool trust**. You might be asked to provide OAuth app client information, choose a Amazon Cognito user pool, choose an IAM IdP, or enter a custom identifier for a developer provider.

   a. Choose the **Role settings** for each IdP. You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With a Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see Assigning precedence values to groups.

      i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

      ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

   b. Configure **Attributes for access control** for each IdP. Attributes for access control maps user claims to principal tags that Amazon Cognito applies to their temporary session. You can build IAM policies to filter user access based on the tags that you apply to their session.

      i. To apply no principal tags, choose **Inactive**.

      ii. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

      iii. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

6. In **Configure properties**, enter a **Name** under **Identity pool name**.

7. Under **Basic (classic) authentication**, choose whether you want to **Activate basic flow**. With the basic flow active, you can bypass the role selections you made for your IdPs and call AssumeRoleWithWebIdentity directly. For more information, see Identity pools authentication flow.

8. Under **Tags**, choose **Add tag** if you want to apply tags to your identity pool.

9. In **Review and create**, confirm the selections that you made for your new identity pool. Select **Edit** to return to the wizard and change any settings. When you're done, select **Create identity pool**.

User IAM roles

An IAM role defines the permissions for your users to access AWS resources, like Amazon Cognito Sync. Users of your application will assume the roles you create. You can specify different roles for authenticated and unauthenticated users. To learn more about IAM roles, see IAM roles.

Authenticated and unauthenticated identities

Amazon Cognito identity pools support both authenticated and unauthenticated identities. Authenticated identities belong to users who are authenticated by any supported identity provider. Unauthenticated identities typically belong to guest users.

- To configure authenticated identities with a public login provider, see Identity pools third-party identity providers.
- To configure your own backend authentication process, see Developer-authenticated identities.

Activate or deactivate guest access

Amazon Cognito identity pools guest access (unauthenticated identities) provides a unique identifier and AWS credentials for users who do not authenticate with an identity provider. If your application allows users who do not log in, you can activate access for unauthenticated identities. To learn more, see Getting started with Amazon Cognito identity pools.

To update guest access in an identity pool

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Guest access**. In an identity pool that doesn't currently support guest access, **Status** is **Inactive**.

    a. If **Guest access** is **Active** and you want to deactivate it, select **Deactivate**.

    b. If **Guest access** is **Inactive** and you want to activate it, select **Edit**.

        - Choose a default IAM role for guest users in your identity pool.

A. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.

B. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see Role trust and permissions.

C. Select **Save changes**.

D. To activate guest access, select **Activate** in the **User access** tab.

Change the role associated with an identity type

Every identity in your identity pool is either authenticated or unauthenticated. Authenticated identities belong to users who are authenticated by a public login provider (Amazon Cognito user pools, Login with Amazon, Sign in with Apple, Facebook, Google, SAML, or any OpenID Connect Providers) or a developer provider (your own backend authentication process). Unauthenticated identities typically belong to guest users.

For each identity type, there is an assigned role. This role has a policy attached to it that dictates which AWS services that role can access. When Amazon Cognito receives a request, the service determines the identity type, determines the role assigned to that identity type, and uses the policy attached to that role to respond. By modifying a policy or assigning a different role to an identity type, you can control which AWS services an identity type can access. To view or modify the policies associated with the roles in your identity pool, see the AWS IAM Console.

To change the identity pool default authenticated or unauthenticated role

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Guest access** or **Authenticated access**. In an identity pool that isn't currently configured for that access type, **Status** is **Inactive**. Select **Edit**.
4. Choose a default IAM role for guest or authenticated users in your identity pool.

a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.

b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see Role trust and permissions.

5. Select **Save changes**.

Edit identity providers

If you allow your users to authenticate using consumer identity providers (for example, Amazon Cognito user pools, Login with Amazon, Sign in with Apple, Facebook, or Google), you can specify your application identifiers in the Amazon Cognito identity pools (federated identities) console. This associates the application ID (provided by the public login provider) with your identity pool.

You can also configure authentication rules for each provider from this page. Each provider allows up to 25 rules. The rules are applied in the order you save for each provider. For more information, see Using role-based access control.

> **Warning**
>
> Changing the linked IdP application ID in your identity pool prevents existing users from authenticating with that identity pool. For more information, see Identity pools third-party identity providers.

To update an identity pool identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.

3. Locate **Identity providers**. Choose the identity provider that you want to edit. If you want to add a new IdP, select **Add identity provider**.

   - If you chose **Add identity provider**, choose one of the **Identity types** that you want to add.

4. To change the application ID, choose **Edit** in **Identity provider information**.
5. To change the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, choose **Edit** in **Role settings**.

   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With a Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see Assigning precedence values to groups.

     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

     ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

6. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, choose **Edit** in **Attributes for access control**.

   a. To apply no principal tags, choose **Inactive**.

   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

7. Select **Save changes**.

Delete an identity pool

You can't undo identity pool deletion. After you delete an identity pool, all apps and users that depend on it stop working.

To delete an identity pool

1. Choose **Identity pools** from the Amazon Cognito console. Select the radio button next to the identity pool that you want to delete.
2. Select **Delete**.
3. Enter or paste the name of your identity pool and select **Delete**.

> **Warning**
>
> When you select the Delete button, you will permanently delete your identity pool and all the user data it contains. Deleting an identity pool will cause applications and other services using the identity pool to stop working.

Delete an identity from an identity pool

When you delete an identity from an identity pool, you remove the identifying information that Amazon Cognito has stored for that federated user. When your user requests credentials again, they receive a new identity ID if your identity pool still trusts their identity provider. You can't undo this operation.

To delete an identity

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the check boxes next to the identities that you want to delete and choose **Delete**. Confirm that you want to delete the identities and choose **Delete**.

Identity pools authentication flow

Amazon Cognito helps you create unique identifiers for your end users that are kept consistent across devices and platforms. Amazon Cognito also delivers temporary, limited-privilege credentials to your application to access AWS resources. This page covers the basics of how authentication in Amazon Cognito works and explains the lifecycle of an identity inside your identity pool.

External provider authflow

A user authenticating with Amazon Cognito goes through a multi-step process to bootstrap their credentials. Amazon Cognito has two different flows for authentication with public providers: enhanced and basic.

Once you complete one of these flows, you can access other AWS services as defined by your role's access policies. By default, the Amazon Cognito console creates roles with access to the Amazon Cognito Sync store and to Amazon Mobile Analytics. For more information on how to grant additional access, see IAM roles.

Identity pools accept the following artifacts from providers:

Provider	Authentication artifact
Amazon Cognito user pool	ID token
OpenID Connect (OIDC)	ID token
SAML 2.0	SAML assertion
Social provider	Access token

The enhanced (simplified) authentication flow

When you use the enhanced authflow, your app first presents a proof of authentication from an authorized Amazon Cognito user pool or third-party identity provider in a GetId request.

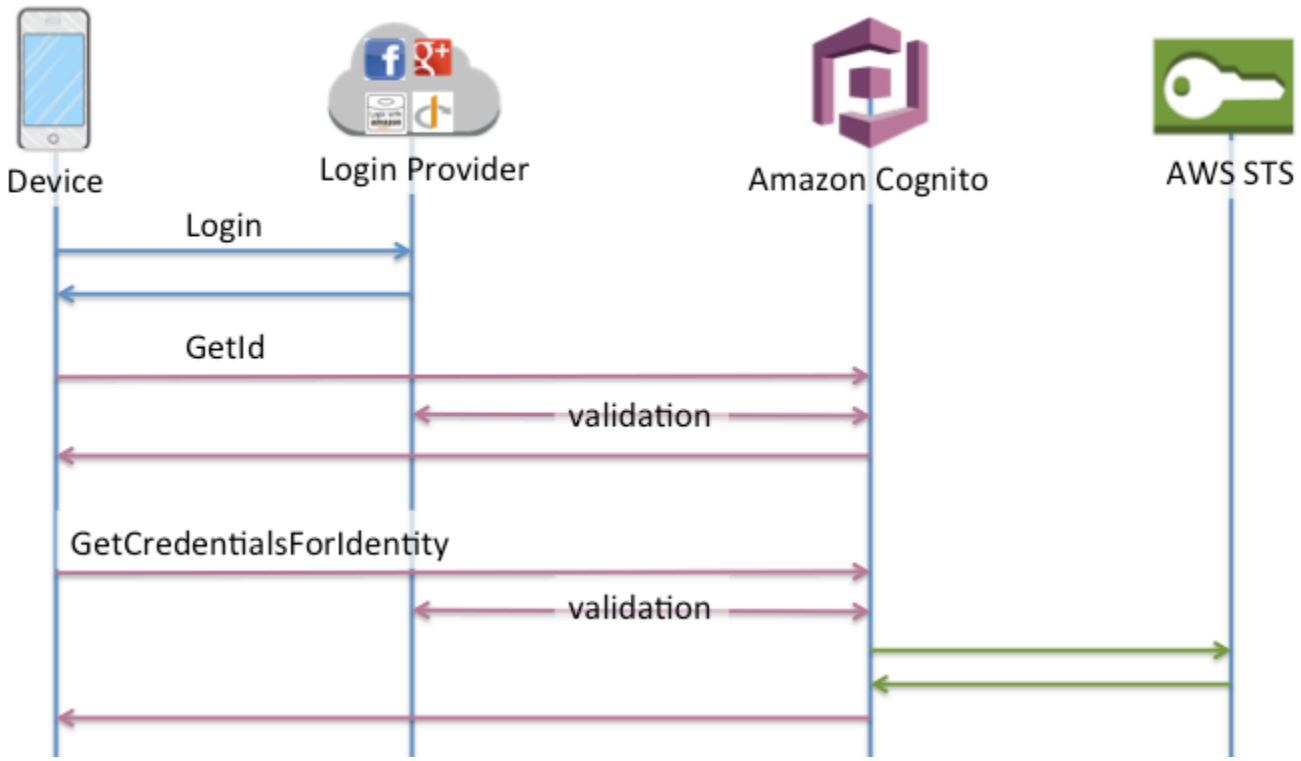
1. Your application presents a proof of authentication–a JSON web token or a SAML assertion–from an authorized Amazon Cognito user pool or third-party identity provider in a GetID request.
2. Your identity pool returns an identity ID.
3. Your application combines the identity ID with the same proof of authentication in a GetCredentialsForIdentity request.
4. Your identity pool returns AWS credentials.
5. Your application signs AWS API requests with the temporary credentials.

Enhanced authentication manages the logic of IAM role selection and credentials retrieval in your identity pool configuration. You can configure your identity pool to select a default role, to

apply attribute-based access control (ABAC) or role-based access control (RBAC) principles to role selection. The AWS credentials from enhanced authentication are valid for one hour.

Order of operations in Enhanced authentication

1. `GetId`
2. `GetCredentialsForIdentity`



The basic (classic) authentication flow

When you implement the basic authentication flow, your application selects the IAM role that you want users to assume.

1. Your application presents a proof of authentication–a JSON web token or a SAML assertion–from an authorized Amazon Cognito user pool or third-party identity provider in a GetID request.
2. Your identity pool returns an identity ID.
3. Your application combines the identity ID with the same proof of authentication in a GetOpenIdToken request.

4. `GetOpenIdToken` returns a new OAuth 2.0 token that is issued by your identity pool.
5. Your application presents the new token in an AssumeRoleWithWebIdentity request.
6. AWS Security Token Service (AWS STS) returns AWS credentials.
7. Your application signs AWS API requests with the temporary credentials.

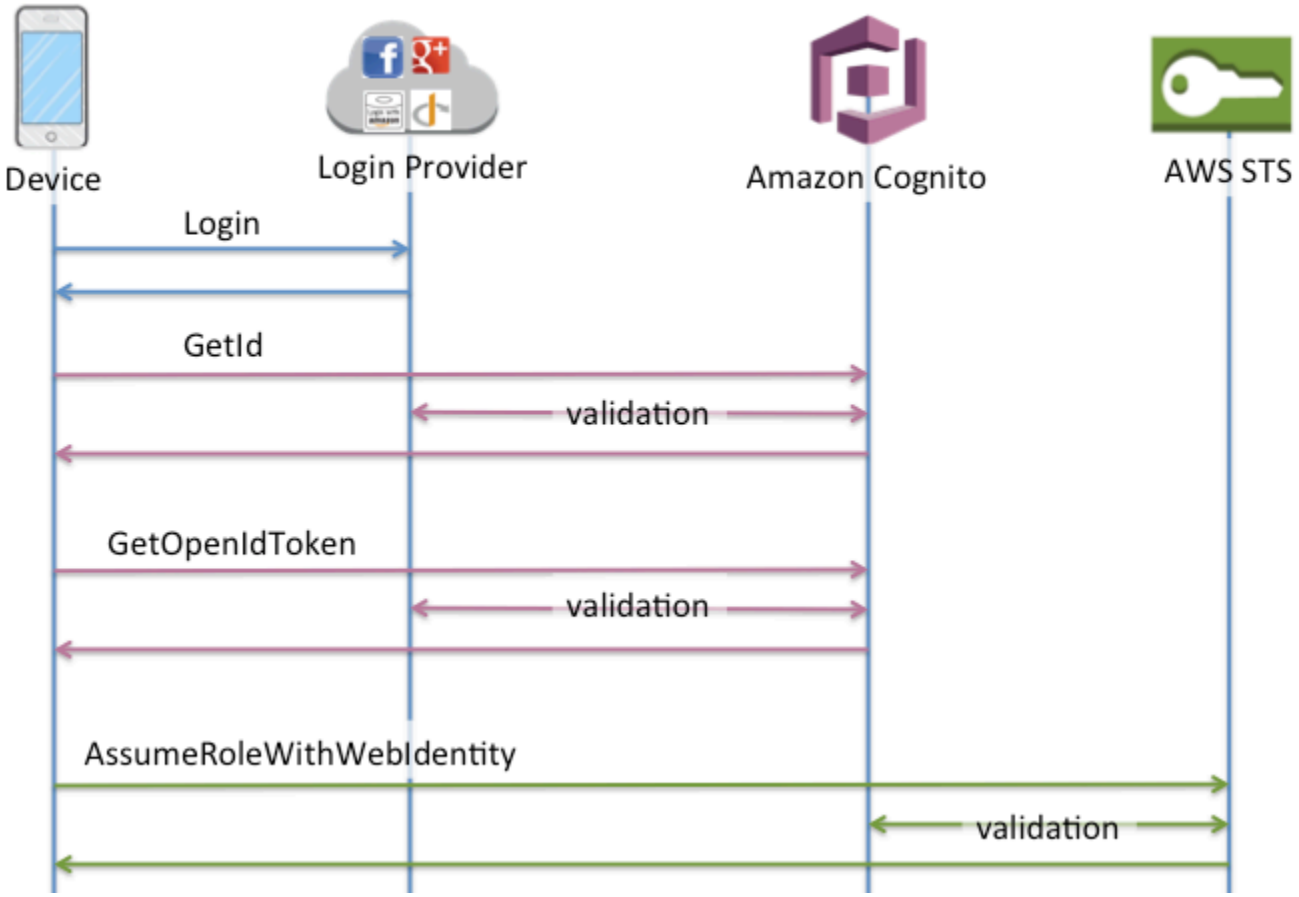
The basic workflow gives you more granular control over the credentials that you distribute to your users. The `GetCredentialsForIdentity` request of the enhanced authflow requests a role based on the contents of an access token. The `AssumeRoleWithWebIdentity` request in the classic workflow grants your app a greater ability to request credentials for any AWS Identity and Access Management role that you have configured with a sufficient trust policy. You can also request a custom role session duration.

You can sign in with the Basic authflow in user pools that don't have role mappings. This type of identity pool doesn't have a default authenticated or unauthenticated role, and doesn't have role-based or attribute-based access control configured. When you attempt `GetOpenIdToken` in an identity pool with role mappings, you receive the following error.

```
Basic (classic) flow is not supported with RoleMappings, please use enhanced flow.
```

Order of operations in Basic authentication

1. `GetId`
2. `GetOpenIdToken`
3. `AssumeRoleWithWebIdentity`

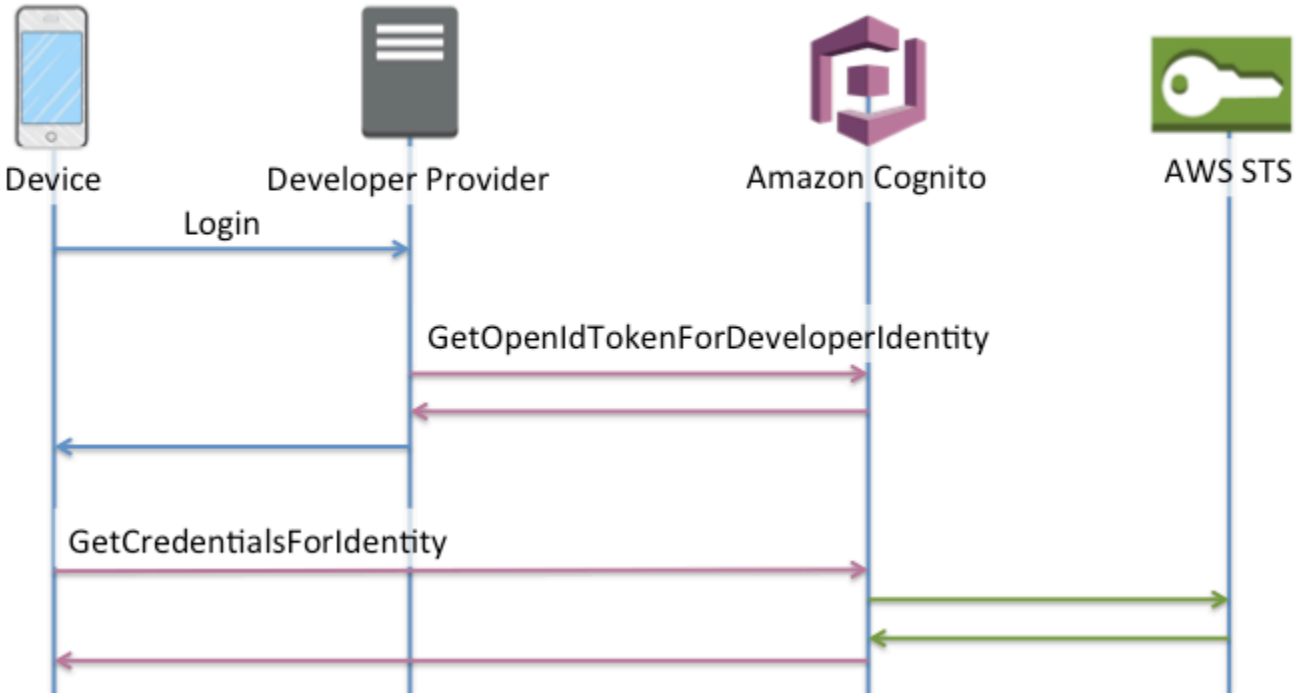


The developer-authenticated authentication flow

When using Developer-authenticated identities, your client uses a different authflow that includes code outside of Amazon Cognito to validate the user in your own authentication system. From the perspective of your identity pool, the claims that you present in your request for an identity are arbitrary identifiers, and the authentication is authorized by IAM credentials that you encode in your application.

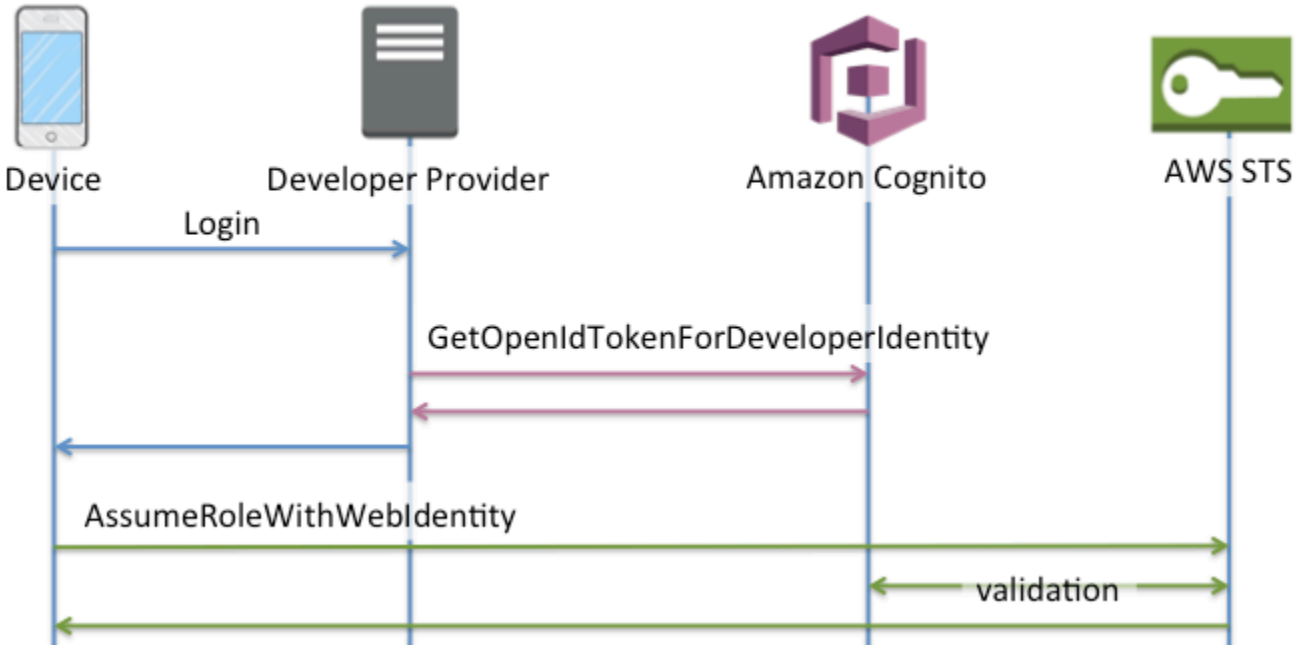
Order of operations in Enhanced authentication with a developer provider

1. Login via Developer Provider (code outside of Amazon Cognito)
2. Validate the user login (code outside of Amazon Cognito)
3. GetOpenIdTokenForDeveloperIdentity
4. GetCredentialsForIdentity



Order of operations in basic authentication with a developer provider

1. Implement logic outside of identity pool to sign in and generate a developer-provider identifier.
2. Retrieve stored server-side AWS credentials.
3. Send developer provider identifier in a GetOpenIdTokenForDeveloperIdentity API request signed with authorized AWS credentials.
4. Request application credentials with AssumeRoleWithWebIdentity.



Which authentication flow should I implement?

The **enhanced flow** is the most secure choice with the lowest level of developer effort:

- The enhanced flow reduces the complexity, size, and rate of API requests.
- Your application doesn't need to make additional API requests to AWS STS.
- Your identity pool evaluates your users for the IAM role credentials that they should receive. You don't need to embed logic for role selection in your client.

> **Important**
>
> When you create a new identity pool, don't activate basic (classic) authentication by default, as a best practice. To implement basic authentication, first evaluate the trust relationships of your IAM roles for web identities. Then build the logic for role selection into your client and secure the client against modification by users.

The **basic authentication flow** delegates the logic of IAM role selection to your application. In this flow, Amazon Cognito validates your user's authenticated or unauthenticated session and issues a token that you can exchange for credentials with AWS STS. Users can exchange the tokens from

basic authentication for any IAM roles that trust your identity pool and `amr`, or authenticated/unauthenticated state.

Similarly, understand that **developer authentication** is a shortcut around validation of identity provider authentication. Amazon Cognito trusts the AWS credentials that authorize a GetOpenIdTokenForDeveloperIdentity request without additional validation of the request contents. Secure the secrets that authorize developer authentication from access by users.

Authentication flow API operations overview

GetId

The GetId API call is the first call necessary to establish a new identity in Amazon Cognito.

Unauthenticated access

Amazon Cognito can grant unauthenticated guest access in your applications. If this feature is enabled in your identity pool, users can request a new identity ID at any time via the `GetId` API. The application is expected to cache this identity ID to make subsequent calls to Amazon Cognito. The AWS Mobile SDKs and the AWS SDK for JavaScript in the Browser have credentials providers that handle this caching for you.

Authenticated access

When you've configured your application with support for a public login provider (Facebook, Google+, Login with Amazon, or Sign in with Apple), users can also supply tokens (OAuth or OpenID Connect) that identify them in those providers. When used in a call to `GetId`, Amazon Cognito creates a new authenticated identity or returns the identity already associated with that particular login. Amazon Cognito does this by validating the token with the provider and making sure of the following:

- The token is valid and from the configured provider.
- The token is not expired.
- The token matches the application identifier created with that provider (for example, Facebook app ID).
- The token matches the user identifier.

GetCredentialsForIdentity

The GetCredentialsForIdentity API can be called after you establish an identity ID. This operation is functionally equivalent to calling GetOpenIdToken, then AssumeRoleWithWebIdentity.

For Amazon Cognito to call `AssumeRoleWithWebIdentity` on your behalf, your identity pool must have IAM roles associated with it. You can do this via the Amazon Cognito console or manually via the SetIdentityPoolRoles operation.

GetOpenIdToken

Make a GetOpenIdToken API request after you establish an identity ID. Cache identity IDs after your first request, and start subsequent basic (classic) sessions for that identity with `GetOpenIdToken`.

The response to a `GetOpenIdToken` API request is a token that Amazon Cognito generates. You can submit this token as the `WebIdentityToken` parameter in an AssumeRoleWithWebIdentity request.

Before you submit the OpenID token, verify it in your app. You can use OIDC libraries in your SDK or a library like aws-jwt-verify to confirm that Amazon Cognito issued the token. The signing key ID, or `kid`, of the OpenID token is one of those listed in the Amazon Cognito Identity jwks_uri document†. These keys are subject to change. Your function that verifies Amazon Cognito Identity tokens should periodically update its list of keys from the *jwks_uri* document. Amazon Cognito sets the refresh duration in the *jwks_uri* cache-control response header, currently set to a `max-age` of 30 days.

Unauthenticated access

To obtain a token for an unauthenticated identity, you only need the identity ID itself. It is not possible to get an unauthenticated token for authenticated identities or identities that you have deactivated.

Authenticated access

If you have an authenticated identity, you must pass at least one valid token for a login already associated with that identity. All tokens passed in during the `GetOpenIdToken` call must pass the same validation mentioned earlier; if any of the tokens fail, the whole call fails. The response from the `GetOpenIdToken` call also includes the identity ID. This is because the identity ID that you pass in may not be the one that is returned.

Linking logins

If you submit a token for a login that is not already associated with any identity, the login is considered to be "linked" to the associated identity. You may only link one login per public provider. Attempts to link more than one login with a public provider results in a

`ResourceConflictException` error response. If a login is merely linked to an existing identity, the identity ID returned from `GetOpenIdToken` is the same as the one that you passed in.

Merging identities

If you pass in a token for a login that is not currently linked to the given identity, but is linked to another identity, the two identities are merged. Once merged, one identity becomes the parent/owner of all associated logins and the other is disabled. In this case, the identity ID of the parent/owner is returned. You must update your local cache if this value differs. The providers in the AWS Mobile SDKs or AWS SDK for JavaScript in the Browser perform this operation for you.

GetOpenIdTokenForDeveloperIdentity

The GetOpenIdTokenForDeveloperIdentity operation replaces the use of GetId and GetOpenIdToken from the device when using developer authenticated identities. Because your application signs requests to this API operation with AWS credentials, Amazon Cognito trusts that the user identifier supplied in the request is valid. Developer authentication replaces the token validation that Amazon Cognito performs with external providers.

The payload for this API includes a `logins` map. This map must contain the key of your developer provider and a value as an identifier for the user in your system. If the user identifier isn't already linked to an existing identity, Amazon Cognito creates a new identity and returns the new identity ID and an OpenID Connect token for that identity. If the user identifier is already linked, Amazon Cognito returns the pre-existing identity ID and an OpenID Connect token. Cache developer identity IDs after your first request, and start subsequent basic (classic) sessions for that identity with `GetOpenIdTokenForDeveloperIdentity`.

The response to a `GetOpenIdTokenForDeveloperIdentity` API request is a token that Amazon Cognito generates. You can submit this token as the `WebIdentityToken` parameter in an `AssumeRoleWithWebIdentity` request.

Before you submit the OpenID Connect token, verify it in your app. You can use OIDC libraries in your SDK or a library like aws-jwt-verify to confirm that Amazon Cognito issued the token. The signing key ID, or `kid`, of the OpenID Connect token is one of those listed in the Amazon Cognito Identity *jwks_uri* document†. These keys are subject to change. Your function that verifies Amazon Cognito Identity tokens should periodically update its list of keys from the *jwks_uri* document. Amazon Cognito sets the refresh duration in the *jwks_uri* `cache-control` response header, currently set to a `max-age` of 30 days.

Linking logins

As with external providers, supplying additional logins that are not already associated with an identity implicitly links those logins to that identity. If you link an external provider login to an identity, the user can use the external provider authflow with that provider. However, they cannot use your developer provider name in the logins map when calling `GetId` or `GetOpenIdToken`.

Merging identities

With developer authenticated identities, Amazon Cognito supports both implicit merging and explicit merging through the MergeDeveloperIdentities API. With explicit merging, you can mark two identities with user identifiers in your system as a single identity. If you supply the source and destination user identifiers, Amazon Cognito merges them. The next time you request an OpenID Connect token for either user identifier, the same identity id is returned.

AssumeRoleWithWebIdentity

After you have an OpenID Connect token, you can then trade this for temporary AWS credentials through the AssumeRoleWithWebIdentity API request to AWS Security Token Service (AWS STS).

Because there is no restriction on the number of identities that you can create, it is important to understand the permissions that you're granting to your users. Set up different IAM roles for your application: one for unauthenticated users, and one for authenticated users. The Amazon Cognito console can create default roles when you first set up your identity pool. These roles have effectively no permissions granted. Modify them to meet your needs.

Learn more about Role trust and permissions.

† The default Amazon Cognito Identity *jwks_uri* document contains information about the keys that sign tokens for identity pools in most AWS Regions. The following Regions have different *jwks_uri* documents.

Amazon Cognito Identity JSON web key URIs in other AWS Regions

AWS Region	Path to *jwks_uri* document
AWS GovCloud (US-West)	`https://cognito-identity.us-gov-west-1.amazonaws.com/.well-known/jwks_uri`

AWS Region	Path to *jwks_uri* document
China (Beijing)	`https://cognito-identity.cn-north-1.amazonaws.com.cn/.well-known/jwks_uri`
Opt-in Regions like Europe (Milan) and Africa (Cape Town)	`https://cognito-identity.` *Region* `.amazonaws.com/.well-known/jwks_uri`

You can also extrapolate the *jwks_uri* from the issuer or `iss` that you receive in the OpenID token from Amazon Cognito. The OIDC-standard discovery endpoint `<issuer>/.well-known/openid-configuration` lists a path to the *jwks_uri* for your token.

IAM roles

While creating an identity pool, you're prompted to update the IAM roles that your users assume. IAM roles work like this: When a user logs in to your app, Amazon Cognito generates temporary AWS credentials for the user. These temporary credentials are associated with a specific IAM role. With the IAM role, you can define a set of permissions to access your AWS resources.

You can specify default IAM roles for authenticated and unauthenticated users. In addition, you can define rules to choose the role for each user based on claims in the user's ID token. For more information, see Using role-based access control.

By default, the Amazon Cognito console creates IAM roles that provide access to Amazon Mobile Analytics and to Amazon Cognito Sync. Alternatively, you can choose to use existing IAM roles.

Modify IAM roles to allow or restrict access to other services. To do so, log in to the IAM Console. Then select **Roles**, and select a role. The policies attached to the selected role are listed in the **Permissions** tab. You can customize an access policy by selecting the corresponding **Manage Policy** link. To learn more about using and defining policies, see Overview of IAM Policies.

> **Note**
>
> As a best practice, define policies that follow the principle of granting *least privilege*. In other words, the policies include only the permissions that users require to perform their tasks. For more information, see Grant Least Privilege in the *IAM User Guide*.

Remember that unauthenticated identities are assumed by users who do not log in to your app. Typically, the permissions that you assign for unauthenticated identities should be more restrictive than those for authenticated identities.

Topics

- Set up a trust policy
- Access policies
- Role trust and permissions

Set up a trust policy

Amazon Cognito uses IAM roles to generate temporary credentials for your application's users. Access to permissions is controlled by a role's trust relationships. Learn more about Role trust and permissions. Amazon Cognito brokers connections between AWS STS and identity pool IdPs.

The token presented to AWS STS is generated by an identity pool, which translates a user pool, social, or OIDC provider token, or a SAML assertion, to its own token. The identity pool token contains an `aud` claim that is the identity pool ID.

Where the `Principal` of an IAM role trust policy is an identity pools service principal like `cognito-identity.amazonaws.com`, you can't create or modify role trust policies such that they permit any identity pool to assume the role. With the identity pool principal, the `Action` element must have a `Condition` that requires that `AssumeRoleWithWebIndentity` be performed only by your identity pools, as specified by a condition key like `cognito-identity.amazonaws.com:aud`. Other condition keys are available, but `aud` is required. If you attempt to save a role trust policy without a condition of this type, IAM returns an error.

For more information about OIDC (web identity) federation keys, see Available keys for AWS OIDC federation.

The following are available OIDC federation condition keys for Amazon Cognito.

`cognito-identity.amazonaws.com:aud`

Restricts the role to operations from one or more identity pools. Amazon Cognito indicates the source identity pool in the `aud` claim in the identity pool token.

`cognito-identity.amazonaws.com:amr`

Restricts the role to either `authenticated` or `unauthenticated` (guest) users. Amazon Cognito indicates the authentication state in the `amr` claim in the identity pool token.

`cognito-identity.amazonaws.com:sub`

Restricts the role to one or more users by UUID. This UUID is the user's identity ID in the identity pool. This value isn't the `sub` value from the user's original identity provider. Amazon Cognito indicates this UUID in the `sub` claim in the identity pool token.

The following example role trust policy allows the federated service principal `cognito-identity.amazonaws.com` to call the AWS STS API `AssumeRoleWithWebIdentity`. The request will only succeed if the identity pool token in the API request has the following claims.

1. An `aud` claim of the identity pool ID `us-west-2:abcdefg-1234-5678-910a-0e8443553f95`.
2. An `amr` claim of `authenticated` that is added when the user has signed in and isn't a guest user.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "cognito-identity.amazonaws.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "cognito-identity.amazonaws.com:aud": "us-
west-2:abcdefg-1234-5678-910a-0e8443553f95"
                },
                "ForAnyValue:StringLike": {
                    "cognito-identity.amazonaws.com:amr": "authenticated"
                }
            }
        }
```

```
    ]
}
```

Trust policies for IAM roles in Basic (Classic) authentication

Summary

Identity pools can only assume roles on behalf of users in the Basic authentication flow when the trust policy of the target role contains the aud condition.

Basic authentication has the same limitation on insecure role trust policies as enhanced authentication: you can't save a role trust policy that fails to limit supported identity pools with an aud condition. This limitation was not enforced when the service launched. Prior to the enforcement of this requirement, you could create role trust policies that had no additional security conditions. After the enforcement of this requirement, AWS STS permits web identities to assume roles that are not secured with conditions, but those roles **can't be modified** without introducing those conditions.

Enhanced-flow authentication requires that the IAM role be in the same AWS account as the identity pool. But in basic authentication, where your application composes the `AssumeRoleWithWebIdentity` request, your application can request to assume a role in a different account. Your cross-account assume role request will fail, though, if the destination role has a legacy trust policy that doesn't enforce the aud condition.

The token that an identity pool issues for an identity contains information about the originating AWS account of the identity pool. When you present an identity pool token in an AssumeRoleWithWebIdentity API request, AWS STS checks to see if the originating identity pool is in the same AWS account as the IAM role. If AWS STS determines that the request is cross-account, it checks to see if the role trust policy has an aud condition. **The assume-role call fails** if no such condition is present in the role trust policy. If the request is not cross-account, AWS STS doesn't enforce this restriction. As a best practice, always apply a condition of this type to the trust policies of your identity pool roles.

The following is an example trust policy that meets the *minimum* requirements for an IAM role for basic authentication with several identity pools. As a best practice, permit only authenticated identities with a `"cognito-identity.amazonaws.com:amr": "authenticated"` condition too.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Federated": "cognito-identity.amazonaws.com"
            },
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Condition": {
                "StringEquals": {
                    "cognito-identity.amazonaws.com:aud": [
                        "us-west-2:abcdefg-1234-5678-910a-0e8443553f95",
                        "us-west-2:hijklmo-5678-9101-112b-0e4221776g96",
                        "us-west-2:pqrstuv-9101-1121-314c-0e2110887h97"
                    ]
                }
            }
        }
    ]
}
```

Additional trust policy conditions

You can use the following trust policy conditions to define the source identity pools, identities, and providers that can assume IAM roles.

> **Note**
>
> Don't implement the aws:SourceIp condition key in trust policies for IAM roles that identity pools assume in the enhanced authentication flow. Because the enhanced flow generates the AssumeRoleWithWebIdentity request on behalf of your application, the source IP of the request will not be the IP of your application client, and the condition will never be satisfied. Network-based condition keys are valid for roles that identity pools assume through *only* the basic flow, which lacks the service-side features of the enhanced flow.

Reuse roles across identity pools

To reuse a role across multiple identity pools, because they share a common permission set, you can include multiple identity pools, like this:

```
"StringEquals": {
    "cognito-identity.amazonaws.com:aud": [
        "us-east-1:12345678-abcd-abcd-abcd-123456790ab",
        "us-east-1:98765432-dcba-dcba-dcba-123456790ab"
    ]
}
```

Limit access to specific identities

To create a policy limited to a specific set of app users, check the value of `cognito-identity.amazonaws.com:sub`:

```
"StringEquals": {
    "cognito-identity.amazonaws.com:aud": "us-east-1:12345678-abcd-abcd-
abcd-123456790ab",
    "cognito-identity.amazonaws.com:sub": [
        "us-east-1:12345678-1234-1234-1234-123456790ab",
        "us-east-1:98765432-1234-1234-1243-123456790ab"
    ]
}
```

Limit access to specific providers

To create a policy limited to users who have logged in with a specific provider (perhaps your own login provider), check the value of `cognito-identity.amazonaws.com:amr`:

```
"ForAnyValue:StringLike": {
    "cognito-identity.amazonaws.com:amr": "login.myprovider.myapp"
}
```

For example, an app that trusts only Facebook would have the following amr clause:

```
"ForAnyValue:StringLike": {
    "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

Access policies

The permissions that you attach to a role apply to all users who assume that role. To partition your users' access, use policy conditions and variables. For more information, see IAM policy elements: Variables and tags. You can use the `sub` condition to restrict actions to Amazon Cognito identity IDs in your access policies. Use this option with caution, particularly for unauthenticated identities, which lack a consistent user ID. For more information about the IAM policy variables for web federation with Amazon Cognito, see IAM and AWS STS condition context keys in the *AWS Identity and Access Management User Guide*.

For additional security protection, Amazon Cognito applies a scope-down policy to credentials that you assign your unauthenticated users in the enhanced flow, using `GetCredentialsForIdentity`. The scope-down policy adds an Inline session policy and an AWS managed session policy to the IAM policies that you apply to your unauthenticated role. Because you must grant access in both the IAM policies for your role and the session policies, the scope-down policy limits users' access to services other than those in the list that follows.

> **Note**
>
> In the basic (classic) flow, you make your own AssumeRoleWithWebIdentity API request, and can apply these restrictions to the request. As a best security practice, don't assign any permissions above this scope-down policy to unauthenticated users.

Amazon Cognito also prevents authenticated and unauthenticated users from making API requests to Amazon Cognito identity pools and Amazon Cognito Sync. Other AWS services might place restrictions on service access from web identities.

In a successful request with the enhanced flow, Amazon Cognito makes an `AssumeRoleWithWebIdentity` API request in the background. Among the parameters in this request, Amazon Cognito includes the following.

1. Your user's identity ID.
2. The ARN of the IAM role that your user wants to assume.
3. A `policy` parameter that adds an *inline session policy*.
4. A `PolicyArns.member.N` parameter whose value is an *AWS managed policy* that grants additional permissions in Amazon CloudWatch.

Services that unauthenticated users can access

When you use the enhanced flow, the scope-down policies that Amazon Cognito applies to your user's session prevent them from using any services other than those listed in the following table. For a subset of services, only specific actions are allowed.

Category	Service
Analytics	Amazon Data Firehose
	Amazon Managed Service for Apache Flink
Application Integration	Amazon Simple Queue Service
AR & VR	Amazon Sumerian[1]
Business Applications	Amazon Mobile Analytics
	Amazon Simple Email Service
Compute	AWS Lambda
Cryptography & PKI	AWS Key Management Service[1]
Database	Amazon DynamoDB
	Amazon SimpleDB
Front-end Web & Mobile	AWS AppSync
	Amazon Location Service
	Amazon Simple Notification Service
	Amazon Pinpoint
	Amazon Location Service
Game Development	Amazon GameLift Servers
Internet of Things (IoT)	AWS IoT

Category	Service
Machine Learning	Amazon CodeWhisperer
	Amazon Comprehend
	Amazon Lex
	Amazon Machine Learning
	Amazon Personalize
	Amazon Polly
	Amazon Rekognition
	Amazon SageMaker AI[1]
	Amazon Textract[1]
	Amazon Transcribe
	Amazon Translate
Management & Governance	Amazon CloudWatch
	Amazon CloudWatch Logs
Networking & Content Delivery	Amazon API Gateway
Security, Identity, & Compliance	Amazon Cognito user pools
Storage	Amazon Simple Storage Service

[1] For the AWS services in the following table, the inline policy grants a subset of actions. The table displays the available actions in each.

AWS service	Maximum permissions for unauthenticated enhanced flow users

AWS Key Management Service	`Encrypt`
	`Decrypt`
	`ReEncryptTo`
	`ReEncryptFrom`
	`GenerateDataKey`
	`GenerateDataKeyPair`
	`GenerateDataKeyPair`
	`GenerateDataKeyPairWithoutPlaintext`
	`GenerateDataKeyWithoutPlaintext`

Amazon SageMaker AI	`InvokeEndpoint`
Amazon Textract	`DetectDocumentText` `AnalyzeDocument`
Amazon Sumerian	`View*`

Amazon Location Service	`SearchPlaceIndex*`
	`GetPlace`
	`CalculateRoute*`
	`*Geofence`
	`*Geofences`
	`*DevicePosition*`

To grant access to AWS services beyond this list, activate the **basic (classic) authentication flow** in your identity pool. If your users see `NotAuthorizedException` errors from AWS services that are allowed by the policies assigned to the IAM role for unauthenticated users, evaluate whether you can remove that service from your use case. If you can't, switch to the basic flow.

The inline session policy for guest users

Amazon Cognito first applies an inline policy in the request for IAM credentials. The inline session policy restricts your user's effective permissions from including access to any AWS services outside those in the following list. You must also grant permissions to these AWS services in the policies that you apply to the user's IAM role. A user's effective permissions for an assumed-role session are the intersection of the policies assigned to their role, and their session policy. For more information, see Session policies in the *AWS Identity and Access Management User Guide*.

Amazon Cognito adds the following inline policy to sessions for your users in AWS Regions that are enabled by default. For an overview of the net effect of the inline policy and other session policies, see Services that unauthenticated users can access.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cloudwatch:*",
                "logs:*",
                "dynamodb:*",
                "kinesis:*",
                "mobileanalytics:*",
                "s3:*",
                "ses:*",
                "sns:*",
                "sqs:*",
                "lambda:*",
                "machinelearning:*",
                "execute-api:*",
                "iot:*",
                "gamelift:*",
                "cognito-identity:*",
```

```
                "cognito-idp:*",
                "lex:*",
                "polly:*",
                "comprehend:*",
                "translate:*",
                "transcribe:*",
                "rekognition:*",
                "mobiletargeting:*",
                "firehose:*",
                "appsync:*",
                "personalize:*",
                "sagemaker:InvokeEndpoint",
                "cognito-sync:*",
                "codewhisperer:*",
                "textract:DetectDocumentText",
                "textract:AnalyzeDocument",
                "sdb:*"
            ],
            "Resource": [
                "*"
            ]
        }
    ]
}
```

For all other Regions, the inline scope-down policy includes everything listed in the default Regions except for the following `Action` statements.

```
            "cognito-sync:*",
            "sumerian:View*",
            "codewhisperer:*",
            "textract:DetectDocumentText",
            "textract:AnalyzeDocument",
            "sdb:*"
```

The AWS managed session policy for guests

Amazon Cognito also applies an AWS managed policy as a session policy to the enhanced-flow sessions of unauthenticated guests. This policy limits the scope of unauthenticated users' permissions with the policy `AmazonCognitoUnAuthedIdentitiesSessionPolicy`.

You must also grant this permission in the policies that you attach to your unauthenticated IAM role. A user's effective permissions for an assumed-role session are the intersection of the IAM policies assigned to their role, and their session policies. For more information, see Session policies in the *AWS Identity and Access Management User Guide*.

For an overview of the net effect of this AWS managed policy and other session policies, see Services that unauthenticated users can access.

The `AmazonCognitoUnAuthedIdentitiesSessionPolicy` managed policy has the following permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [{
        "Effect": "Allow",
        "Action": [
            "rum:PutRumEvents",
            "polly:*",
            "comprehend:*",
            "translate:*",
            "transcribe:*",
            "rekognition:*",
            "mobiletargeting:*",
            "firehose:*",
            "personalize:*",
            "sagemaker:InvokeEndpoint",
            "geo:GetMap*",
            "geo:SearchPlaceIndex*",
            "geo:GetPlace",
            "geo:CalculateRoute*",
            "geo:*Geofence",
            "geo:*Geofences",
            "geo:*DevicePosition*",
            "kms:Encrypt",
            "kms:Decrypt",
            "kms:ReEncryptTo",
            "kms:ReEncryptFrom",
            "kms:GenerateDataKey",
            "kms:GenerateDataKeyPair",
            "kms:GenerateDataKeyPairWithoutPlaintext",
```

```
            "kms:GenerateDataKeyWithoutPlaintext"
        ],
        "Resource": "*"
    }]
}
```

Access policy examples

In this section, you can find example Amazon Cognito access policies that grant your users the minimum permissions necessary to do specific operation. You can further limit the permissions for a given identity ID by using policy variables where possible. For example, using ${cognito-identity.amazonaws.com:sub}. For more information, see Understanding Amazon Cognito Authentication Part 3: Roles and Policies on the *AWS Mobile Blog*.

> **Note**
>
> As a security best practice, policies should include only the permissions that users require to perform their tasks. This means that you should try to always scope access to an individual identity for objects whenever possible.

Grant an identity read access to a single object in Amazon S3

The following access policy grants read permissions to an identity to retrieve a single object from a given S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws-eusc:s3:::amzn-s3-demo-bucket/assets/my_picture.jpg"]
    }
  ]
```

```
}
```

Grant an identity both read and write access to identity specific paths in Amazon S3

The following access policy grants read and write permissions to access a specific prefix "folder" in an S3 bucket by mapping the prefix to the `${cognito-identity.amazonaws.com:sub}` variable.

With this policy, an identity such as `us-east-1:12345678-1234-1234-1234-123456790ab` inserted via `${cognito-identity.amazonaws.com:sub}` can get, put, and list objects into `arn:aws:s3:::amzn-s3-demo-bucket/us-east-1:12345678-1234-1234-1234-123456790ab`. However, the identity would not be granted access to other objects in `arn:aws:s3:::amzn-s3-demo-bucket`.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": ["s3:ListBucket"],
      "Effect": "Allow",
      "Resource": ["arn:aws-eusc:s3:::amzn-s3-demo-bucket"],
      "Condition": {"StringLike": {"s3:prefix": ["${cognito-
identity.amazonaws.com:sub}/*"]}}
    },
    {
      "Action": [
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws-eusc:s3:::amzn-s3-demo-bucket/${cognito-
identity.amazonaws.com:sub}/*"]
    }
  ]
}
```

A similar access model is achieved with Amazon S3 Access Grants.

Assign identities fine-grained access to Amazon DynamoDB

The following access policy provides fine-grained access control to DynamoDB resources using Amazon Cognito environment variables. These variables grant access to items in DynamoDB by identity ID. For more information, see Using IAM Policy Conditions for Fine-Grained Access Control in the *Amazon DynamoDB Developer Guide*.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Action": [
        "dynamodb:GetItem",
        "dynamodb:BatchGetItem",
        "dynamodb:Query",
        "dynamodb:PutItem",
        "dynamodb:UpdateItem",
        "dynamodb:DeleteItem",
        "dynamodb:BatchWriteItem"
      ],
      "Resource": [
        "arn:aws-eusc:dynamodb:us-west-2:123456789012:table/MyTable"
      ],
      "Condition": {
        "ForAllValues:StringEquals": {
          "dynamodb:LeadingKeys":  ["${cognito-identity.amazonaws.com:sub}"]
        }
      }
    }
  ]
}
```

Grant an identity permission to invoke a Lambda function

The following access policy grants an identity permission to invoke a Lambda function.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
         {
             "Effect": "Allow",
             "Action": "lambda:InvokeFunction",
             "Resource": [
                 "arn:aws-eusc:lambda:us-west-2:123456789012:function:MyFunction"
             ]
         }
     ]
 }
```

Grant an identity permission to publish records to Kinesis Data Streams

The following access policy allows an identity to use the `PutRecord` operation with any of the Kinesis Data Streams. It can be applied to users that need to add data records to all streams in an account. For more information, see Controlling Access to Amazon Kinesis Data Streams Resources Using IAM in the *Amazon Kinesis Data Streams Developer Guide*.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "kinesis:PutRecord",
            "Resource": [
                "arn:aws-eusc:kinesis:eusc-de-east-1:111122223333:stream/stream1"
            ]
        }
    ]
}
```

Grant an identity access to their data in the Amazon Cognito Sync store

The following access policy grants an identity permissions to access only their own data in the Amazon Cognito Sync store.

JSON

```
{
  "Version":"2012-10-17",
  "Statement":[{
      "Effect":"Allow",
      "Action":"cognito-sync:*",
      "Resource":["arn:aws-eusc:cognito-sync:eusc-de-
east-1:123456789012:identitypool/${cognito-identity.amazonaws.com:aud}/identity/
${cognito-identity.amazonaws.com:sub}/*"]
      }]
  }
```

Role trust and permissions

The way these roles differ is in their trust relationships. The following is an example trust policy for an unauthenticated role:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "eusc-de-east-1:12345678-corner-
cafe-123456790ab"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "unauthenticated"
```

```
        }
      }
    }
  ]
}
```

This policy grants federated users from `cognito-identity.amazonaws.com` (the issuer of the OpenID Connect token) permission to assume this role. Additionally, the policy restricts the `aud` of the token, in this case the identity pool ID, to match the identity pool. Finally, the policy specifies that one of the array members of the multi-value `amr` claim of the token issued by the Amazon Cognito `GetOpenIdToken` API operation has the value `unauthenticated`.

When Amazon Cognito creates a token, it sets the `amr` of the token as either `unauthenticated` or `authenticated`. If `amr` is `authenticated`, the token includes any providers used during authentication. This means that you can create a role that trusts only users that logged in via Facebook by changing the `amr` condition as shown:

```
"ForAnyValue:StringLike": {
  "cognito-identity.amazonaws.com:amr": "graph.facebook.com"
}
```

Be careful when changing your trust relationships on your roles, or when trying to use roles across identity pools. If you don't configure your role correctly to trust your identity pool, an exception from STS results, like the following:

```
AccessDenied -- Not authorized to perform sts:AssumeRoleWithWebIdentity
```

If you see this message, check that your identity pool and authentication type have an appropriate role.

Security best practices for Amazon Cognito identity pools

Amazon Cognito identity pools provide temporary AWS credentials for your application. AWS accounts often contain both the resources that your application users need, and private back-end resources. The IAM roles and policies that make up AWS credentials can grant access to any of these resources.

The primary best practice of identity pool configuration is to ensure that your application can get the job done without excess or unintended privilege. To guard against security misconfiguration,

review these recommendations before the launch of each application that you want to release to production.

Topics

- IAM configuration best practices
- Identity pool configuration best practices

IAM configuration best practices

When a guest or authenticated user initiates a session in your application that requires identity pool credentials, your application retrieves temporary AWS credentials for an IAM role. The credentials might be for a default role, a role chosen by rules in your identity pool configuration, or for a custom role chosen by your app. With the permissions assigned to each role, your user gains access to your AWS resources.

For more information about general IAM best practices, see IAM best practices in the AWS Identity and Access Management User Guide.

Use trust policy conditions in IAM roles

IAM requires that roles for identity pools have at least one trust policy condition. This condition can, for example, set the role's scope to authenticated users only. AWS STS also requires that cross-account basic authentication requests have two specific conditions: `cognito-identity.amazonaws.com:aud` and `cognito-identity.amazonaws.com:amr`. As a best practice, apply both of these conditions in all IAM roles that trust the identity pools service principal `cognito-identity.amazonaws.com`.

- `cognito-identity.amazonaws.com:aud`: The *aud* claim in the identity pool token must match a trusted identity pool ID.
- `cognito-identity.amazonaws.com:amr`: The *amr* claim in the identity pool token must be either *authenticated* or *unauthenticated*. With this condition, you can reserve access to a role only to unauthenticated guests, or only to authenticated users. You can further refine the value of this condition to restrict the role to users from a specific provider, for example `graph.facebook.com`.

The following example role trust policy grants access to a role under the following conditions:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "eusc-de-
east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
      }
    }
  ]
}
```

Elements that relate to identity pools

- `"Federated": "cognito-identity.amazonaws.com"`: Users must come from an identity pool.
- `"cognito-identity.amazonaws.com:aud": "us-east-1:a1b2c3d4-5678-90ab-cdef-example11111"`: Users must come from the specific identity pool `us-east-1:a1b2c3d4-5678-90ab-cdef-example11111`.
- `"cognito-identity.amazonaws.com:amr": "authenticated"`: Users must be authenticated. Guest users can't assume the role.

Apply least privilege permissions

When you set permissions with IAM policies for authenticated access or guest access, grant only the specific permissions required to perform specific tasks, or *least privilege* permissions. The following

example IAM policy, when applied to a role, grants read-only access to a single image file in an Amazon S3 bucket.

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Action": [
        "s3:GetObject"
      ],
      "Effect": "Allow",
      "Resource": ["arn:aws-eusc:s3:::mybucket/assets/my_picture.jpg"]
    }
  ]
}
```

Identity pool configuration best practices

Identity pools have flexible options for the generation of AWS credentials. Don't take design shortcuts when your application can work with the most secure methods.

Understand the effects of guest access

Unauthenticated guest access permits users to retrieve data from your AWS account before they sign in. Anyone who knows your identity pool ID can request unauthenticated credentials. Your identity pool ID isn't confidential information. When you activate guest access, the AWS permissions that you grant to unauthenticated sessions are available to everyone.

As a best practice, leave guest access deactivated and fetch required resources only after users authenticate. If your application requires access to resources before sign-in, take the following precautions.

- Familiarize yourself with the automatic limitations placed on unauthenticated roles.
- Monitor and adjust the permissions of your unauthenticated IAM roles to match the specific needs of your application.
- Grant access to specific resources.
- Secure the trust policy of your default unauthenticated IAM role.

- Activate guest access only when you are confident that you would grant the permissions in your IAM role to anyone on the internet.

Use enhanced authentication by default

With basic (classic) authentication, Amazon Cognito delegates selection of the IAM role to your app. In contrast, the enhanced flow uses the centralized logic in your identity pool to determine the IAM role. It also provides additional security for unauthenticated identities with a scope-down policy that sets an upper limit on IAM permissions. The enhanced flow is the most secure choice with the lowest level of developer effort. To learn more about these options, see Identity pools authentication flow .

The basic flow can expose the client-side logic that goes into role selection and assembly of the AWS STS API request for credentials. The enhanced flow hides both the logic and the assume-role request behind identity pool automation.

When you configure basic authentication, apply IAM best practices to your IAM roles and their permissions.

Use developer providers securely

Developer authenticated identities are a feature of identity pools for server-side applications. The only evidence of authentication that identity pools require for developer authentication are the AWS credentials of an identity pool developer. Identity pools don't enforce any restrictions on the validity of the developer-provider identifiers that you present in this authentication flow.

As a best practice, only implement developer providers under the following conditions:

- To create accountability for the use of developer-authenticated credentials, design your developer provider name and identifiers to indicate the authentication source. For example: `"Logins" : {"MyCorp provider" : "[provider application ID]"}`.
- Avoid long-lived user credentials. Configure your server-side client to request identities with service-linked roles like EC2 instance profiles and Lambda execution roles.
- Avoid mixing internal and external sources of trust in the same identity pool. Add your developer provider and your single sign-on (SSO) providers in separate identity pools.

Using attributes for access control

Attributes for access control is the Amazon Cognito identity pools implementation of attribute-based access control (ABAC). You can use IAM policies to control access to AWS resources through Amazon Cognito identity pools based on user attributes. These attributes can be drawn from social and corporate identity providers. You can map attributes within providers' access and ID tokens or SAML assertions to tags that can be referenced in the IAM permissions policies.

You can choose default mappings or create your own custom mappings in Amazon Cognito identity pools. The default mappings allow you to write IAM policies based on a fixed set of user attributes. Custom mappings allow you to select a custom set of user attributes that are referenced in the IAM permissions policies. The **Attribute names** in the Amazon Cognito console are mapped to **Tag key for principal**, which are the tags that are referenced in the IAM permissions policy.

For example, let's say that you own a media streaming service with a free and a paid membership. You store the media files in Amazon S3 and tag them with free or premium tags. You can use attributes for access control to allow access to free and paid content based on user membership level, which is part of the user's profile. You can map the membership attribute to a tag key for principal to be passed on to the IAM permissions policy. This way you can create a single permissions policy and conditionally allow access to premium content based on the value of membership level and tag on the content files.

Topics

- Using attributes for access control with Amazon Cognito identity pools
- Using attributes for access control policy example
- Turn off attributes for access control (console)
- Default provider mappings

Using attributes to control access has several benefits:

- Permissions management is more efficient when you use attributes for access control. You can create a basic permissions policy that uses user attributes instead of creating multiple policies for different job functions.
- You don't need to update your policies whenever you add or remove resources or users for your application. The permissions policy will only grant the access to users with the matching user attributes. For example, you might need to control the access to certain S3 buckets based on the job title of users. In that case, you can create a permissions policy to allow access to these

files only for users within the defined job title. For more information, see IAM Tutorial: Use SAML session tags for ABAC.

- Attributes can be passed as principal tags to a policy that allows or denies permissions based on the values of those attributes.

Using attributes for access control with Amazon Cognito identity pools

Before you can use attributes for access control, ensure that you meet the following prerequisites:

- An AWS account
- User pool
- Identity pool
- Set up an SDK
- Integrated identity providers
- Credentials

To use attributes for access control, the **Claim** that you set as the source of data sets the value of the **Tag Key** that you choose. Amazon Cognito applies the tag key and value to your user's session. Your IAM policies can evaluate your user's access from the `${aws:PrincipalTag/tagkey}` condition. IAM evaluates the value of your user's tag against the policy.

You must prepare IAM roles whose credentials you want to pass to your users. The trust policy of these roles must permit Amazon Cognito to assume the role for your user. For attributes for access control, you must also allow Amazon Cognito to apply principal tags to your user's temporary session. Grant permission to assume the role with the action AssumeRoleWithWebIdentity. Grant permission to tag users' sessions with the permission-only action `sts:TagSession`. For more information, see Passing session tags in AWS Security Token Service in the *AWS Identity and Access Management User Guide*. For an example trust policy that grants `sts:AssumeRoleWithWebIdentity` and `sts:TagSession` permissions to the Amazon Cognito service principal `cognito-identity.amazonaws.com`, see Using attributes for access control policy example.

> **Important**
>
> If you map custom attributes from a Amazon Cognito user pool to principal tags, ensure that end users cannot modify those attributes. An attribute that a user can change through

`UpdateUserAttributes` should not serve as a security boundary in your IAM policies. To prevent user modification, do one of the following:

- Remove the attribute from the WriteAttributes of every app client in your user pool, not only the app client that you configured in your identity pool.
- Define the attribute as `Mutable: false` when you add it to your user pool schema.

User attributes are shared by all app clients in a user pool. If a user changes an attribute through any app client, the new value appears in that user's ID tokens from every app client.
When you don't specify `WriteAttributes` for an app client, the app client has write access to all standard and custom attributes. Specify `WriteAttributes` for each app client that you create, including app clients that you add later.
Users can't change an immutable attribute after their user profile is created. However, they can set its value when they sign up. If your user pool allows self-service sign-up, remove the attribute from the `WriteAttributes` of every app client. Then do one of the following:

- Set the value when you create users with `AdminCreateUser`.
- Reject sign-up requests that contain unexpected values. For more information, see the section called “Pre sign-up”.

Additionally, ensure that attribute values mapped to principal tags do not contain characters that have special meaning in IAM policy evaluation, such as * (asterisk) or ? (question mark). In IAM policy `Resource` elements and `StringLike` condition operators, these characters function as wildcards. If you set an attribute value to * and map it to a principal tag, the `${aws:PrincipalTag/`*`tagkey`*`}` evaluation might match more resources than intended. This can grant access to resources belonging to other tenants or projects. Validate attribute values in your application logic. Alternatively, use a pre token generation Lambda trigger to reject or sanitize values that contain these characters before your application passes them as principal tags.
If your application requires the attribute to remain user-writable for other purposes, use a pre token generation Lambda trigger to validate or override the claim before it is included in the ID token.

To configure attributes for access control in the console

1. Sign in to the Amazon Cognito console and select **Identity pools**. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Identity providers**. Choose the identity provider that you want to edit. If you want to add a new IdP, select **Add identity provider**.
4. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, choose **Edit** in **Attributes for access control**.
   a. To apply no principal tags, choose **Inactive**.
   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.
   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
5. Select **Save changes**.

Using attributes for access control policy example

Consider a scenario where an employee from the legal department of a company needs to list all files in buckets that belong to their department and are classified with their security level. Assume the token that this employee gets from the identity provider contains the following claims.

Claims

```
{
            "...": "other claims",
            "sub" : "57e7b692-4f66-480d-98b8-45a6729b4c88",
            "department" : "legal",
            "clearance" : "confidential",
            "...": "other claims"
        }
```

These attributes can be mapped to tags and referenced in IAM permissions policies as principal tags. You can now manage access by changing the user profile on the identity provider's end. Alternatively, you can change attributes on the resource side by using names or tags without changing the policy itself.

The following permissions policy does two things:

- Allows list access to all S3 buckets that end with a prefix that matches the user's department name.
- Allows read access on files in these buckets as long as the clearance tag on the file matches user's clearance attribute.

Permissions policy

JSON

```
   {
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": "s3:List*",
            "Resource": "arn:aws-eusc:s3:::*-${aws:PrincipalTag/department}"
        },
        {
            "Effect": "Allow",
            "Action": "s3:GetObject*",
            "Resource": "arn:aws-eusc:s3:::*-${aws:PrincipalTag/department}/*",
            "Condition": {
                "StringEquals": {
                    "s3:ExistingObjectTag/clearance": "${aws:PrincipalTag/
clearance}"
                }
            }
        }
    ]
}
```

The trust policy determines who can assume this role. The trust relationship policy allows the use of `sts:AssumeRoleWithWebIdentity` and `sts:TagSession` to allow access. It adds conditions to restrict the policy to the identity pool that you created and it makes sure that it's for an authenticated role.

Trust policy

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": [
        "sts:AssumeRoleWithWebIdentity",
        "sts:TagSession"
      ],
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "IDENTITY-POOL-ID"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
      }
    }
  ]
}
```

Turn off attributes for access control (console)

Follow this procedure to deactivate attributes for access control.

To deactivate attributes for access control in the console

1. Sign in to the Amazon Cognito console and select **Identity pools**. Select an identity pool.
2. Choose the **User access** tab.
3. Locate **Identity providers**. Choose the identity provider that you want to edit.
4. Choose **Edit** in **Attributes for access control**.
5. To apply no principal tags, choose **Inactive**.

6. Select **Save changes**.

Default provider mappings

The following table has the default mapping information for the authentication providers that Amazon Cognito supports.

Provider	Token type	Principal tag values	Example
Amazon Cognito user pool	ID token	aud(client ID) and sub(user ID)	"6jk8ltokc7ac9es6jrtg9q572f", "57e7b692-4f66-480d-98b8-45a6729b4c88"
Facebook	Access token	aud(app_id), sub(user_id)	"492844718097981", "112177216992379"
Google	ID token	aud(client ID) and sub(user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.apps.googleusercontent.com", "109220063452404746097"
SAML	Assertions	"http://schemas.xmlsoap.org/ws/2005/05/identity/claims/nameidentifier" , "http://schemas.xmlsoap.org/ws/2005/05/identity/claims/name"	"auth0|5e28d196f8f55a0eaaa95de3", "user123@gmail.com"
Apple	ID token	aud(client ID) and sub (user ID)	"com.amazonaws.ec2-54-80-172-243.compute-1.client",

Provider	Token type	Principal tag values	Example
			"001968.a6ca34e9c1e742458a26cf8005854be9.0733"
Amazon	Access token	aud (Client ID on Amzn Dev Ac), user_id(user ID)	"amzn1.application-oa2-client.9d70d9382d3446108aaee3dd763a0fa6", "amzn1.account.AGHNIFJQMFSBG3G6XCPVB35ORQAA"
Standard OIDC providers	ID and access tokens	aud (as client_id), sub (as user ID)	"620493171733-eebk7c0hcp5lj3e1tlqp1gntt3k0rncv.apps.googleusercontent.com", "109220063452404746097"
Twitter	Access token	aud (app ID; app Secret), sub (user ID)	"DfwifTtKEX1FiIBRnOTlR0CFK;Xgj5xb8xIrIVCPjXgLIdkW7fXmwcJJrFvnoK9gwZkLexo1y5z1", "1269003884292222976"
DevAuth	Map	Not applicable	"tag1", "tag2"

> **Note**
>
> The default attribute mappings option is automatically populated for the **Tag Key for Principal** and **Attribute** names. You can't change default mappings.

Using role-based access control

Amazon Cognito identity pools assign your authenticated users a set of temporary, limited-privilege credentials to access your AWS resources. The permissions for each user are controlled through IAM roles that you create. You can define rules to choose the role for each user based on claims in the user's ID token. You can define a default role for authenticated users. You can also define a separate IAM role with limited permissions for guest users who are not authenticated.

Creating roles for role mapping

It is important to add the appropriate trust policy for each role so that it can only be assumed by Amazon Cognito for authenticated users in your identity pool. Here is an example of such a trust policy:

JSON

```
{
  "Version":"2012-10-17",
  "Statement": [
    {
      "Sid": "",
      "Effect": "Allow",
      "Principal": {
        "Federated": "cognito-identity.amazonaws.com"
      },
      "Action": "sts:AssumeRoleWithWebIdentity",
      "Condition": {
        "StringEquals": {
          "cognito-identity.amazonaws.com:aud": "eusc-de-east-1:12345678-corner-
cafe-123456790ab"
        },
        "ForAnyValue:StringLike": {
          "cognito-identity.amazonaws.com:amr": "authenticated"
        }
```

```
      }
    }
  ]
}
```

This policy allows federated users from `cognito-identity.amazonaws.com` (the issuer of the OpenID Connect token) to assume this role. Additionally, the policy restricts the `aud` of the token, in this case the identity pool ID, to match the identity pool. Finally, the policy specifies that one of the array members of the multi-value `amr` claim of the token issued by the Amazon Cognito `GetOpenIdToken` API action has the value `authenticated`.

Granting pass-role permission

To allow a user to set roles with permissions in excess of the user's existing permissions on an identity pool, grant them `iam:PassRole` permission to pass the role to the `set-identity-pool-roles` API. For example, if the user cannot write to Amazon S3, but the IAM role that the user sets on the identity pool grants write permission to Amazon S3, the user can only set this role if `iam:PassRole` permission is granted for the role. The following example policy shows how to allow `iam:PassRole` permission.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "Stmt1",
            "Effect": "Allow",
            "Action": [
                "iam:PassRole"
            ],
            "Resource": [
                "arn:aws-eusc:iam::123456789012:role/myS3WriteAccessRole"
            ]
        }
    ]
}
```

In this policy example, the `iam:PassRole` permission is granted for the `myS3WriteAccessRole` role. The role is specified using the role's Amazon Resource Name (ARN). You must also attach this policy to your user. For more information, see Working with Managed Policies.

> **Note**
>
> Lambda functions use resource-based policy, where the policy is attached directly to the Lambda function itself. When creating a rule that invokes a Lambda function, you do not pass a role, so the user creating the rule does not need the `iam:PassRole` permission. For more information about Lambda function authorization, see Manage Permissions: Using a Lambda Function Policy.

Using tokens to assign roles to users

For users who log in through Amazon Cognito user pools, roles can be passed in the ID token that was assigned by the user pool. The roles appear in the following claims in the ID token:

- The `cognito:preferred_role` claim is the role ARN.
- The `cognito:roles` claim is a comma-separated string containing a set of allowed role ARNs.

The claims are set as follows:

- The `cognito:preferred_role` claim is set to the role from the group with the best (lowest) `Precedence` value. If there is only one allowed role, `cognito:preferred_role` is set to that role. If there are multiple roles and no single role has the best precedence, this claim is not set.
- The `cognito:roles` claim is set if there is at least one role.

When using tokens to assign roles, if there are multiple roles that can be assigned to the user, Amazon Cognito identity pools (federated identities) chooses the role as follows:

- Use the GetCredentialsForIdentity `CustomRoleArn` parameter if it is set and it matches a role in the `cognito:roles` claim. If this parameter doesn't match a role in `cognito:roles`, deny access.
- If the `cognito:preferred_role` claim is set, use it.
- If the `cognito:preferred_role` claim is not set, the `cognito:roles` claim is set, and `CustomRoleArn` is not specified in the call to `GetCredentialsForIdentity`, then the

Role resolution setting in the console or the `AmbiguousRoleResolution` field (in the `RoleMappings` parameter of the SetIdentityPoolRoles API) is used to determine the role to be assigned.

Using rule-based mapping to assign roles to users

Rules allow you to map claims from an identity provider token to IAM roles.

Each rule specifies a token claim (such as a user attribute in the ID token from an Amazon Cognito user pool), match type, a value, and an IAM role. The match type can be `Equals`, `NotEqual`, `StartsWith`, or `Contains`. If a user has a matching value for the claim, the user can assume that role when the user gets credentials. For example, you can create a rule that assigns a specific IAM role for users with a `custom:dept` custom attribute value of `Sales`.

> **Note**
>
> In the rule settings, custom attributes require the `custom:` prefix to distinguish them from standard attributes.

Rules are evaluated in order, and the IAM role for the first matching rule is used, unless `CustomRoleArn` is specified to override the order. For more information about user attributes in Amazon Cognito user pools, see Working with user attributes.

You can set multiple rules for an authentication provider in the identity pool (federated identities) console. Rules are applied in order. You can drag the rules to change their order. The first matching rule takes precedence. If the match type is `NotEqual` and the claim doesn't exist, the rule is not evaluated. If no rules match, the **Role resolution** setting is applied to either **Use default authenticated role** or **Deny request**.

In the API and CLI, you can specify the role to be assigned when no rules match in the `AmbiguousRoleResolution` field of the RoleMapping type, which is specified in the `RoleMappings` parameter of the SetIdentityPoolRoles API.

To add rule-based mapping to an identity provider in the Amazon Cognito console, add or update an IdP and select **Choose role with rules** under **Role selection**. From there, you can add rules that map provider claims to IAM roles.

You can set up rule-based mapping for identity providers in the AWS CLI or API with the `RulesConfiguration` field of the RoleMapping type. You can specify this field in the `RoleMappings` parameter of the SetIdentityPoolRoles API.

For example, the following AWS CLI command adds a rule that assigns the role `arn:aws:iam::123456789012:role/Sacramento_team_S3_admin` to users in your Sacramento location who were authenticated by OIDC IdP `arn:aws:iam::123456789012:oidc-provider/myOIDCIdP`:

```
aws cognito-identity set-identity-pool-roles --region us-east-1 --cli-input-json
 file://role-mapping.json
```

Contents of `role-mapping.json`:

```
{
    "IdentityPoolId": "us-east-1:12345678-corner-cafe-123456790ab",
    "Roles": {
        "authenticated": "arn:aws:iam::123456789012:role/myS3WriteAccessRole",
        "unauthenticated": "arn:aws:iam::123456789012:role/myS3ReadAccessRole"
    },
    "RoleMappings": {
        "arn:aws:iam::123456789012:oidc-provider/myOIDCIdP": {
            "Type": "Rules",
            "AmbiguousRoleResolution": "AuthenticatedRole",
            "RulesConfiguration": {
                "Rules": [
                    {
                        "Claim": "locale",
                        "MatchType": "Equals",
                        "Value": "Sacramento",
                        "RoleARN": "arn:aws:iam::123456789012:role/
Sacramento_team_S3_admin"
                    }
                ]
            }
        }
    }
}
```

For each user pool or other authentication provider that you configure for an identity pool, you can create up to 25 rules. This limit is not adjustable. For more information, see Quotas in Amazon Cognito.

Token claims to use in rule-based mapping

Amazon Cognito

An Amazon Cognito ID token is represented as a JSON Web Token (JWT). The token contains claims about the identity of the authenticated user, such as `name`, `family_name`, and `phone_number`. For more information about standard claims, see the OpenID Connect specification. Apart from standard claims, the following are the additional claims specific to Amazon Cognito:

- `cognito:groups`
- `cognito:roles`
- `cognito:preferred_role`

Amazon

The following claims, along with possible values for those claims, can be used with Login with Amazon:

- `iss`: www.amazon.com
- aud: App Id
- `sub`: `sub` from the Login with Amazon token

Facebook

The following claims, along with possible values for those claims, can be used with Facebook:

- `iss`: graph.facebook.com
- aud: App Id
- `sub`: `sub` from the Facebook token

Google

A Google token contains standard claims from the OpenID Connect specification. All of the claims in the OpenID token are available for rule-based mapping. See Google's OpenID Connect site to learn about the claims available from the Google token.

Apple

An Apple token contains standard claims from the OpenID Connect specification. See Authenticating Users with Sign in with Apple in Apple's documentation to learn more about the claim available from the Apple token. Apple's token doesn't always contain `email`.

OpenID

All of the claims in the Open Id token are available for rule-based mapping. For more information about standard claims, see the OpenID Connect specification. Refer to your OpenID provider documentation to learn about any additional claims that are available.

SAML

Claims are parsed from the received SAML assertion. All the claims that are available in the SAML assertion can be used in rule-based mapping.

Best practices for role-based access control

> **Important**
>
> If the claim that you are mapping to a role can be modified by the end user, any end user can assume your role and set the policy accordingly. Only map claims that cannot be directly set by the end user to roles with elevated permissions. In an Amazon Cognito user pool, you can set per-app read and write permissions for each user attribute.

> **Important**
>
> If you set roles for groups in an Amazon Cognito user pool, those roles are passed through the user's ID token. To use these roles, you must also set **Choose role from token** for the authenticated role selection for the identity pool.
> You can use the **Role resolution** setting in the console and the `RoleMappings` parameter of the SetIdentityPoolRoles API to specify what the default behavior is when the correct role cannot be determined from the token.

Getting credentials

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. This section describes how to get credentials and how to retrieve an Amazon Cognito identity from an identity pool.

Amazon Cognito supports both authenticated and unauthenticated identities. Unauthenticated users do not have their identity verified, making this role appropriate for guest users of your app or in cases when it doesn't matter if users have their identities verified. Authenticated users log in to your application through a third-party identity provider, or a user pool, that verifies their identities. Make sure you scope the permissions of resources appropriately so you don't grant access to them from unauthenticated users.

Amazon Cognito identities are not credentials. They are exchanged for credentials using web identity federation support in the AWS Security Token Service (AWS STS). To obtain AWS credentials for your app users, use your AWS SDK's Cognito identity pool credential provider. For example, in the AWS SDK for JavaScript (v3), use the `fromCognitoIdentityPool` function from the `@aws-sdk/credential-providers` package. The identity is then exchanged for credentials using AWS STS.

Web identity credentials providers are part of the default credential provider chain in AWS SDKs. To set your identity pool token in a local `config` file for an AWS SDK or the AWS CLI, add a `web_identity_token_file` profile entry. See Assume role credential provider in the AWS SDKs and Tools Reference Guide.

To learn more about how to populate web identity credentials in your SDK, refer to the SDK developer guide. For best results, start your project with the identity pool integration that's built in to AWS Amplify.

AWS SDK resources for getting and setting credentials with identity pools

- Identity Pool Federation (Android) in the Amplify Dev Center
- Identity Pool Federation (iOS) in the Amplify Dev Center
- Using Amazon Cognito Identity to authenticate users in the AWS SDK for JavaScript Developer Guide
- Amazon Cognito credentials provider in the AWS SDK for .NET Developer Guide
- Specify Credentials Programmatically in the AWS SDK for Go Developer Guide

- Supply temporary credentials in code in the AWS SDK for Java 2.x Developer Guide
- assumeRoleWithWebIdentityCredentialProvider provider in the AWS SDK for PHP Developer Guide
- Assume Role With Web Identity Provider in the AWS SDK for Python (Boto3) documentation
- Specifying your credentials and default region in the AWS SDK for Rust Developer Guide

The following sections provide example code in some legacy AWS SDKs.

Android

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

To use a Amazon Cognito identity pool in an Android app, set up AWS Amplify. For more information, see Authentication in the *Amplify Dev Center*.

Retrieving an Amazon Cognito identity

If you're allowing unauthenticated users, you can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately. If you're authenticating users, you can retrieve the identity ID after you've set the login tokens in the credentials provider:

```
String identityId = credentialsProvider.getIdentityId();
Log.d("LogTag", "my ID is " + identityId);
```

Note

Do not call `getIdentityId()`, `refresh()`, or `getCredentials()` in the main thread of your application. As of Android 3.0 (API Level 11), your app will automatically fail and throw a NetworkOnMainThreadException if you perform network I/O on the main application thread. You must move your code to a background thread using `AsyncTask`. For more information, consult the Android documentation. You can also call `getCachedIdentityId()` to retrieve an ID, but only if one is already cached locally. Otherwise, the method will return null.

iOS - Objective-C

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito identity pools support both authenticated and unauthenticated identities. To provide AWS credentials to your app, complete the following steps.

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see Swift Authentication and Flutter Authentication in the *Amplify Dev Center*.

Retrieving an Amazon Cognito identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
// Retrieve your Amazon Cognito ID
[[credentialsProvider getIdentityId] continueWithBlock:^id(AWSTask *task) {
    if (task.error) {
        NSLog(@"Error: %@", task.error);
    }
    else {
        // the task result will contain the identity id
        NSString *cognitoId = task.result;
    }
    return nil;
}];
```

Note

`getIdentityId` is an asynchronous call. If an identity ID is already set on your provider, you can call `credentialsProvider.identityId` to retrieve that identity, which is cached locally. However, if an identity ID is not set on your provider, calling `credentialsProvider.identityId` will return `nil`. For more information, consult the Amplify iOS SDK reference.

iOS - Swift

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see Swift Authentication in the *Amplify Dev Center*.

Retrieving an Amazon Cognito identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
// Retrieve your Amazon Cognito ID
credentialsProvider.getIdentityId().continueWith(block: { (task) -> AnyObject? in
    if (task.error != nil) {
        print("Error: " + task.error!.localizedDescription)
    }
    else {
        // the task result will contain the identity id
        let cognitoId = task.result!
        print("Cognito id: \(cognitoId)")
    }
    return task;
})
```

> **Note**
>
> `getIdentityId` is an asynchronous call. If an identity ID is already set on your provider, you can call `credentialsProvider.identityId` to retrieve that identity, which is cached locally. However, if an identity ID is not set on your provider, calling `credentialsProvider.identityId` will return `nil`. For more information, consult the Amplify iOS SDK reference.

JavaScript

If you have not yet created one, create an identity pool in the Amazon Cognito console before using `fromCognitoIdentityPool`.

After you configure an identity pool with your identity providers, you can use the `fromCognitoIdentityPool` function from the `@aws-sdk/credential-providers` package to authenticate users. Rather than setting a global `AWS.config.credentials`, attach the credentials provider directly to each service client through its `credentials` property. The v3 SDK resolves the provider lazily the first time the client makes a request, so there is no separate call to obtain credentials. The following example attaches the provider to an `S3Client`:

```
import { S3Client } from "@aws-sdk/client-s3";
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

// Set the region where your identity pool exists (for example, us-east-1 or eu-west-1)
const region = "REGION";

// Attach the credentials provider to your service client
const client = new S3Client({
  region,
  credentials: fromCognitoIdentityPool({
    identityPoolId: "IDENTITY_POOL_ID",
    logins: { // optional tokens, used for authenticated login
      "graph.facebook.com": "FBTOKEN",
      "www.amazon.com": "AMAZONTOKEN",
      "accounts.google.com": "GOOGLETOKEN",
      "appleid.apple.com": "APPLETOKEN"
    },
    clientConfig: { region },
  }),
});

// Credentials are resolved lazily when the client first makes a request.
// To inspect them directly, call the provider and await the result
// (from inside an async function):
const credentials = await client.config.credentials();
const { accessKeyId, secretAccessKey, sessionToken } = credentials;
```

The optional `logins` property is a map of identity provider names to the identity tokens for those providers. How you get the token from your identity provider depends on the provider you use. For example, if Facebook is one of your identity providers, you might use the `FB.login` function from the Facebook SDK to get an identity provider token:

```
FB.login(function (response) {
    if (response.authResponse) { // logged in
```

```
        const region = "us-east-1";
        const client = new S3Client({
          region,
          credentials: fromCognitoIdentityPool({
            identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
            logins: {
              "graph.facebook.com": response.authResponse.accessToken
            },
            clientConfig: { region },
          }),
        });

        console.log('You are now logged in.');
    } else {
        console.log('There was a problem logging you in.');
    }
});
```

Retrieving an Amazon Cognito identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user. If you allow unauthenticated users, you can retrieve it immediately. If you authenticate users, retrieve it after you set the login tokens. The `fromCognitoIdentityPool` provider exposes the `identityId` on the credentials that it resolves, so you can read it from the resolved credentials of a client that uses the provider:

```
// Inside an async function:
const { identityId } = await client.config.credentials();
```

Unity

You can use Amazon Cognito to deliver temporary, limited-privilege credentials to your application, so that your users can access AWS resources. Amazon Cognito supports both authenticated and unauthenticated identities. To provide AWS credentials to your app, follow the steps below.

The AWS SDK for Unity is now part of the SDK for .NET. To get started with Amazon Cognito in the SDK for .NET, see Amazon Cognito credentials provider in the AWS SDK for .NET Developer Guide. Or see Amplify Dev Center for options for building an app with AWS Amplify.

Retrieving an Amazon Cognito identity

You can retrieve a unique Amazon Cognito identifier (identity ID) for your end user immediately if you're allowing unauthenticated users or after you've set the login tokens in the credentials provider if you're authenticating users:

```
string identityId = await credentials.GetIdentityIdAsync();
```

Accessing AWS services with temporary credentials

The result of a successful authentication with an identity pool is a set of AWS credentials. With these credentials, your application can make requests to AWS resources that are protected with IAM authentication. With the various AWS SDKs that you can add to your applications to access identity pools API operations, you can make unauthenticated API requests that produce temporary credentials. Then you can add SDKs for other AWS services to your client and sign requests with those temporary credentials. The IAM permissions granted to your temporary-credentials role must permit the operations that you request from other services.

After you configure your Amazon Cognito credentials provider and retrieve AWS credentials, create an AWS service client. The following are some examples from AWS SDK documentation.

AWS SDK resources for creating a client

- AWS Client configuration in the AWS SDK for C++ Developer Guide
- Using the AWS SDK for Go V2 with AWS services in the AWS SDK for Go Developer Guide
- Configuring HTTP clients in the AWS SDK for Java 2.x Developer Guide
- Creating and calling service objects in the AWS SDK for JavaScript Developer Guide
- Creating clients in the AWS SDK for Python (Boto3) documentation
- Creating a service client in the AWS SDK for Rust Developer Guide
- Using clients in the AWS SDK for Swift Developer Guide

The following snippet initializes an Amazon DynamoDB client:

Android

To use a Amazon Cognito identity pool in an Android app, set up AWS Amplify. For more information, see Authentication in the *Amplify Dev Center*.

```
// Create a service client with the provider
```

```
AmazonDynamoDB client = new AmazonDynamoDBClient(credentialsProvider);
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

iOS - Objective-C

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see Swift Authentication and Flutter Authentication in the *Amplify Dev Center*.

```
// create a configuration that uses the provider
AWSServiceConfiguration *configuration = [AWSServiceConfiguration
 configurationWithRegion:AWSRegionUSEast1 provider:credentialsProvider];
// get a client with the default service configuration
AWSDynamoDB *dynamoDB = [AWSDynamoDB defaultDynamoDB];
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

iOS - Swift

To use a Amazon Cognito identity pool in an iOS app, set up AWS Amplify. For more information, see Swift Authentication in the *Amplify Dev Center*.

```
// get a client with the default service configuration
let dynamoDB = AWSDynamoDB.default()

// get a client with a custom configuration
AWSDynamoDB.register(with: configuration!, forKey: "USWest2DynamoDB");
let dynamoDBCustom = AWSDynamoDB(forKey: "USWest2DynamoDB")
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

JavaScript

```
// Create a service client with the provider
var dynamodb = new AWS.DynamoDB({region: 'us-west-2'});
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited-privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

Unity

The AWS SDK for Unity is now part of the SDK for .NET. To get started with Amazon Cognito in the SDK for .NET, see Amazon Cognito credentials provider in the AWS SDK for .NET Developer Guide. Or see Amplify Dev Center for options for building an app with AWS Amplify.

```
// create a service client that uses credentials provided by Cognito
AmazonDynamoDBClient client = new AmazonDynamoDBClient(credentials, REGION);
```

The credentials provider communicates with Amazon Cognito, retrieving both the unique identifier for authenticated and unauthenticated users as well as temporary, limited-privilege AWS credentials for the AWS Mobile SDK. The retrieved credentials are valid for one hour, and the provider refreshes them when they expire.

Identity pools third-party identity providers

With Amazon Cognito identity pools, you can integrate with a variety of external identity providers (IdPs) to provide temporary AWS credentials through federated authentication in your application. By configuring your identity pool to work with these external IdPs, you can authorize access to back-end AWS resources for your users with authentication by Amazon Cognito user pools, social providers, OIDC providers, or SAML providers. This section covers the steps to set up and integrate IdPs with your Amazon Cognito identity pool.

Using the `logins` property, you can set credentials received from an identity provider (IdP). You can also associate an identity pool with multiple IdPs. For example, you can set both the Facebook and Google tokens in the `logins` property to associate the unique Amazon Cognito identity with both IdP logins. The user can authenticate with either account, but Amazon Cognito returns the same user identifier.

The following instructions guide you through authentication with the IdPs that Amazon Cognito identity pools support.

Topics

- Setting up Facebook as an identity pools IdP
- Setting up Login with Amazon as an identity pools IdP
- Setting up Google as an identity pool IdP
- Setting up Sign in with Apple as an identity pool IdP
- Setting up an OIDC provider as an identity pool IdP
- Setting up a SAML provider as an identity pool IdP

Setting up Facebook as an identity pools IdP

Amazon Cognito identity pools work with Facebook to provide federated authentication for your application users. This section explains how to register and set up your application with Facebook as an IdP.

Set up Facebook

Register your application with Facebook before you authenticate Facebook users and interact with Facebook APIs.

The Facebook Developers portal helps you to set up your application. Do this procedure before you integrate Facebook in your Amazon Cognito identity pool:

> **Note**
>
> Amazon Cognito identity pools federation isn't compatible with Facebook Limited Login. For more information about how to set up Facebook Login for iOS without exceeding the permissions set for Limited Login, see Facebook Login for iOS - Quickstart at *Meta for Developers*.

Setting up Facebook

1. At the Facebook Developers portal, log in with your Facebook credentials.
2. From the **Apps** menu, select **Add a New App**.
3. Select a platform and complete the quick start process.

Android

For more information about how to integrate Android apps with Facebook Login, see the Facebook Getting Started Guide.

iOS - Objective-C

For more information about how to integrate iOS Objective-C apps with Facebook Login, see the Facebook Getting Started Guide.

iOS - Swift

For more information about how to integrate iOS Swift apps with Facebook Login, see the Facebook Getting Started Guide.

JavaScript

For more information about how to integrate JavaScript web apps with Facebook Login, see the Facebook Getting Started Guide.

Configure an identity provider in the Amazon Cognito identity pools console

Use the following procedure to configure your identity provider.

To add a Facebook identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Facebook**.
5. Enter the **App ID** of the OAuth project that you created at Meta for Developers. For more information, see Facebook Login in the *Meta for Developers Docs*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that

will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

   a. To apply no principal tags, choose **Inactive**.

   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Using Facebook

Android

To add Facebook authentication, first follow the Facebook guide and integrate the Facebook SDK into your application. Then add a **Login with Facebook** button to your Android user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

After you authenticate your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

Facebook SDK 4.0 or later:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", AccessToken.getCurrentAccessToken().getToken());
credentialsProvider.setLogins(logins);
```

Facebook SDK before 4.0:

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("graph.facebook.com", Session.getActiveSession().getAccessToken());
```

```
credentialsProvider.setLogins(logins);
```

The Facebook login process initializes a singleton session in its SDK. The Facebook session object contains an OAuth token that Amazon Cognito uses to generate AWS credentials for your authenticated end user. Amazon Cognito also uses the token to check against your user database for the existence of a user that matches this particular Facebook identity. If the user already exists, the API returns the existing identifier. Otherwise, the API returns a new identifier. The client SDK automatically caches identifiers on the local device.

> **Note**
>
> After you set the logins map, make a call to `refresh` or `get` to retrieve the AWS credentials.

iOS - Objective-C

To add Facebook authentication, first follow the Facebook guide and integrate the Facebook SDK into your application. Then add a Login with Facebook button to your user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user and bind them to a unique Amazon Cognito identity pools (federated identities).

To provide the Facebook access token to Amazon Cognito, implement the AWSIdentityProviderManager protocol.

When you implement the `logins` method, return a dictionary that contains `AWSIdentityProviderFacebook`. This dictionary acts as the key, and the current access token from the authenticated Facebook user acts as the value, as shown in the following code example.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    FBSDKAccessToken* fbToken = [FBSDKAccessToken currentAccessToken];
    if(fbToken){
        NSString *token = fbToken.tokenString;
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook : token }];
    }else{
            return [AWSTask taskWithError:[NSError errorWithDomain:@"Facebook Login"
                                                              code:-1
                                                          userInfo:@{@"error":@"No current
 Facebook access token"}]];
```

```
    }
}
```

When you instantiate the `AWSCognitoCredentialsProvider`, pass the class that implements `AWSIdentityProviderManager` as the value of `identityProviderManager` in the constructor. For more information, go to the AWSCognitoCredentialsProvider reference page and choose **initWithRegionType:identityPoolId:identityProviderManager**.

iOS - Swift

To add Facebook authentication, first follow the Facebook guide and integrate the Facebook SDK into your application. Then add a Login with Facebook button to your user interface. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user and bind them to a unique Amazon Cognito identity pools (federated identities).

> **Note**
>
> Amazon Cognito identity pools federation isn't compatible with Facebook Limited Login. For more information about how to set up Facebook Login for iOS without exceeding the permissions set for Limited Login, see Facebook Login for iOS - Quickstart at *Meta for Developers*.

To provide the Facebook access token to Amazon Cognito, implement the AWSIdentityProviderManager protocol.

When you implement the `logins` method, return a dictionary containing `AWSIdentityProviderFacebook`. This dictionary acts as the key, and the current access token from the authenticated Facebook user acts as the value, as shown in the following code example.

```
class FacebookProvider: NSObject, AWSIdentityProviderManager {
    func logins() -> AWSTask<NSDictionary> {
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }
}
```

When you instantiate the `AWSCognitoCredentialsProvider`, pass the class that implements `AWSIdentityProviderManager` as the value of `identityProviderManager` in the constructor. For more information, go to the AWSCognitoCredentialsProvider reference page and choose **initWithRegionType:identityPoolId:identityProviderManager**.

JavaScript

To add Facebook authentication, follow the Facebook Login for the Web and add the **Login with Facebook** button on your website. The Facebook SDK uses a session object to track its state. Amazon Cognito uses the access token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

After you authenticate your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider.

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { S3Client } from "@aws-sdk/client-s3";

FB.login(function (response) {

  // Check if the user logged in successfully.
  if (response.authResponse) {

    console.log('You are now logged in.');

    // Attach a Cognito identity pool credentials provider to your service client,
    // passing the Facebook access token in the logins map.
    const client = new S3Client({
      region: 'REGION',
      credentials: fromCognitoIdentityPool({
        identityPoolId: 'IDENTITY_POOL_ID',
        logins: {
          'graph.facebook.com': response.authResponse.accessToken
        },
        clientConfig: { region: 'REGION' }
      })
    });

    // The provider resolves and caches AWS credentials the first time the
    // client makes a request. Access AWS resources with client here.

  } else {
    console.log('There was a problem logging you in.');
```

```
  }

});
```

The Facebook SDK obtains an OAuth token that Amazon Cognito uses to generate AWS credentials for your authenticated end user. Amazon Cognito also uses the token to check against your user database for the existence of a user matching this particular Facebook identity. If the user already exists, the API returns the existing identifier. Otherwise a new identifier is returned. Identifiers are automatically cached by the client SDK on the local device.

> **Note**
>
> The credentials provider resolves AWS credentials lazily the first time your service client makes a request. For more information about using Amazon Cognito identity pools with the AWS SDK for JavaScript, see AWS SDK for JavaScript v3 Developer Guide.

Unity

To add Facebook authentication, first follow the Facebook guide and integrate the Facebook SDK into your application. Amazon Cognito uses the Facebook access token from the FB object to generate a unique user identifier that is associated with an Amazon Cognito identity.

After you authenticate your user with the Facebook SDK, add the session token to the Amazon Cognito credentials provider:

```
void Start()
{
    FB.Init(delegate() {
        if (FB.IsLoggedIn) { //User already logged in from a previous session
            AddFacebookTokenToCognito();
        } else {
            FB.Login ("email", FacebookLoginCallback);
        }
    });
}

void FacebookLoginCallback(FBResult result)
{
    if (FB.IsLoggedIn)
    {
```

```
        AddFacebookTokenToCognito();
    }
    else
    {
        Debug.Log("FB Login error");
    }
}

void AddFacebookTokenToCognito()
{
    credentials.AddLogin ("graph.facebook.com",
 AccessToken.CurrentAccessToken.TokenString);
}
```

Before you use `FB.AccessToken`, call `FB.Login()` and make sure `FB.IsLoggedIn` is true.

Setting up Login with Amazon as an identity pools IdP

Amazon Cognito identity pools work with Login with Amazon to provide federated authentication for your mobile and web app users. This section explains how to register and set up your application with Login with Amazon as an identity provider (IdP).

Set up Login with Amazon to work with Amazon Cognito in the Developer Portal. For more information, see Setting Up Login with Amazon in the Login with Amazon FAQ.

> **Note**
>
> You can't natively integrate Login with Amazon on the Unity platform. Instead, use a web view and go through the browser sign-in flow.

Setting up Login with Amazon

Implement Login with Amazon

In the Amazon developer portal, you can set up an OAuth application to integrate with your identity pool, find Login with Amazon documentation, and download SDKs. Choose **Developer console**, then **Login with Amazon** in the developer portal. You can create a security profile for your application and then build Login with Amazon authentication mechanisms into your app. See Getting credentials for more information about how to integrate Login with Amazon authentication with your app.

Amazon issues an OAuth 2.0 **client ID** for your new security profile. You can find the **client ID** on the security profile **Web Settings** tab. Enter the **Security Profile ID** in the **App ID** field of the Login with Amazon IdP in your identity pool.

> **Note**
>
> You enter the **Security Profile ID** in the **App ID** field of the Login with Amazon IdP in your identity pool. This differs from user pools, which use **client ID**.

Configure the external provider in the Amazon Cognito console

To add a Login with Amazon identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Login with Amazon**.
5. Enter the **App ID** of the OAuth project that you created at Login with Amazon. For more information, see Login with Amazon Documentation.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
     ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
   a. To apply no principal tags, choose **Inactive**.

b. To apply principal tags based on sub and aud claims, choose **Use default mappings**.

c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Use Login with Amazon: Android

After you authenticate Amazon login, you can pass the token to the Amazon Cognito credentials provider in the onSuccess method of the TokenListener interface. The code looks like this:

```
@Override
public void onSuccess(Bundle response) {
    String token = response.getString(AuthzConstants.BUNDLE_KEY.TOKEN.val);
    Map<String, String> logins = new HashMap<String, String>();
    logins.put("www.amazon.com", token);
    credentialsProvider.setLogins(logins);
}
```

Use Login with Amazon: iOS - Objective-C

After you authenticate Amazon login, you can pass the token to the Amazon Cognito credentials provider in the requestDidSucceed method of the AMZNAccessTokenDelegate:

```
- (void)requestDidSucceed:(APIResult \*)apiResult {
    if (apiResult.api == kAPIAuthorizeUser) {
        [AIMobileLib getAccessTokenForScopes:[NSArray arrayWithObject:@"profile"]
 withOverrideParams:nil delegate:self];
    }
    else if (apiResult.api == kAPIGetAccessToken) {
        credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyLoginWithAmazon):
 apiResult.result };
    }
}}
```

Use Login with Amazon: iOS - Swift

After you authenticate Amazon login, you can pass the token to the Amazon Cognito credentials provider in the `requestDidSucceed` method of the `AMZNAccessTokenDelegate`:

```
func requestDidSucceed(apiResult: APIResult!) {
    if apiResult.api == API.AuthorizeUser {
        AIMobileLib.getAccessTokenForScopes(["profile"], withOverrideParams: nil,
 delegate: self)
    } else if apiResult.api == API.GetAccessToken {
        credentialsProvider.logins =
 [AWSCognitoLoginProviderKey.LoginWithAmazon.rawValue: apiResult.result]
    }
}
```

Use Login with Amazon: JavaScript

After the user authenticates with Login with Amazon and is redirected back to your website, the Login with Amazon access_token is provided in the query string. Pass that token into the credentials login map.

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({
   region: 'REGION',
   credentials: fromCognitoIdentityPool({
       identityPoolId: 'IDENTITY_POOL_ID',
       logins: {
           'www.amazon.com': 'Amazon Access Token'
       },
       clientConfig: { region: 'REGION' }
   })
});
```

Setting up Google as an identity pool IdP

Amazon Cognito identity pools work with Google to provide federated authentication for your mobile application users. This section explains how to register and set up your application with Google as an IdP.

Android

> **Note**
>
> If your app uses Google and is available on multiple mobile platforms, you should configure it as an OpenID Connect Provider. Add all created client IDs as additional audience values

for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Setting up Google

To activate Google Sign-in for Android, create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **Android** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and then choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see Creating and managing projects in the Google Cloud documentation.

For more information about how to integrate Google into your Android app, see Authenticate users with Sign in with Google in the Google Identity documentation.

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at Google Cloud Platform. For more information, see Setting up OAuth 2.0 in *Google Cloud Platform Console Help*.

6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

     ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

   a. To apply no principal tags, choose **Inactive**.

   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Use Google

To enable login with Google in your application, follow the instructions in the Google documentation for Android. When a user signs in, they request an OpenID Connect authentication token from Google. Amazon Cognito then uses the token to authenticate the user and generate a unique identifier.

The following example code shows how to retrieve the authentication token from the Google Play service:

```
GooglePlayServicesUtil.isGooglePlayServicesAvailable(getApplicationContext());
AccountManager am = AccountManager.get(this);
Account[] accounts = am.getAccountsByType(GoogleAuthUtil.GOOGLE_ACCOUNT_TYPE);
String token = GoogleAuthUtil.getToken(getApplicationContext(), accounts[0].name,
        "audience:server:client_id:YOUR_GOOGLE_CLIENT_ID");
```

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("accounts.google.com", token);
credentialsProvider.setLogins(logins);
```

iOS - Objective-C

> **Note**
>
> If your app uses Google and is available on multiple mobile platforms, configure Google as an OpenID Connect Provider. Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Setting up Google

To enable Google Sign-in for iOS, create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **iOS** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account. Choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see Creating and managing projects in the Google Cloud documentation.

For more information about how to integrate Google into your iOS app, see Google Sign-In for iOS in the Google Identity documentation.

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at Google Cloud Platform. For more information, see Setting up OAuth 2.0 in *Google Cloud Platform Console Help*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

     ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

   a. To apply no principal tags, choose **Inactive**.

   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Use Google

To enable login with Google in your application, follow the Google documentation for iOS. Successful authentication results in an OpenID Connect authentication token, which Amazon Cognito uses to authenticate the user and generate a unique identifier.

Successful authentication results in a `GTMOAuth2Authentication` object, which contains an `id_token`, which Amazon Cognito uses to authenticate the user and generate a unique identifier:

```
- (void)finishedWithAuth: (GTMOAuth2Authentication *)auth error: (NSError *) error {
        NSString *idToken = [auth.parameters objectForKey:@"id_token"];
        credentialsProvider.logins = @{ @(AWSCognitoLoginProviderKeyGoogle): idToken };
    }
```

iOS - Swift

> **Note**
>
> If your app uses Google and is available on multiple mobile platforms, configure Google as an OpenID Connect Provider. Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Setting up Google

To enable Google Sign-in for iOS, create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks for their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **iOS** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see Creating and managing projects in the Google Cloud documentation.

For more information about how to integrate Google into your iOS app, see Google Sign-In for iOS in the Google Identity documentation.

Choose **Manage Identity Pools** from the Amazon Cognito Console home page:

Configuring the external provider in the Amazon Cognito Console

1. Choose the name of the identity pool where you want to enable Google as an external provider. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, choose **Edit identity pool**. The Edit identity pool page appears.
3. Scroll down and choose **Authentication providers** to expand the section.
4. Choose the **Google** tab.
5. Choose **Unlock**.
6. Enter the Google Client ID that you obtained from Google, and then choose **Save Changes**.

Use Google

To enable login with Google in your application, follow the Google documentation for iOS. Successful authentication results in an OpenID Connect authentication token that Amazon Cognito uses to authenticate the user and generate a unique identifier.

Successful authentication results in a `GTMOAuth2Authentication` object that contains an `id_token`. Amazon Cognito uses this token to authenticate the user and generate a unique identifier:

```
func finishedWithAuth(auth: GTMOAuth2Authentication!, error: NSError!) {
    if error != nil {
      print(error.localizedDescription)
    }
    else {
      let idToken = auth.parameters.objectForKey("id_token")
      credentialsProvider.logins = [AWSCognitoLoginProviderKey.Google.rawValue:
 idToken!]
    }
}
```

JavaScript

> **Note**
>
> If your app uses Google and is available on multiple mobile platforms, you should configure Google as an OpenID Connect Provider. Add all created client IDs as additional audience values for better integration. To learn more about Google's cross-client identity model, see Cross-client Identity.

Setting up Google

To enable Google Sign-in for a JavaScript web app, create a Google Developers console project for your application.

1. Go to the Google Developers console and create a new project.
2. Choose **APIs & Services**, then **OAuth consent screen**. Customize the information that Google shows to your users when Google asks their consent to share their profile data with your app.
3. Choose **Credentials**, then **Create credentials**. Choose **OAuth client ID**. Select **Web application** as the **Application type**. Create a separate client ID for each platform where you develop your app.
4. From **Credentials**, choose **Manage service accounts**. Choose **Create service account**. Enter your service account details, and choose **Create and continue**.
5. Grant the service account access to your project. Grant users access to the service account as your app requires.
6. Choose your new service account, choose the **Keys** tab, and **Add key**. Create and download a new JSON key.

For more information about how to use the Google Developers console, see Creating and managing projects in the Google Cloud documentation.

For more information about how to integrate Google into your web app, see Sign in With Google in the Google Identity documentation.

Configure the External Provider in the Amazon Cognito Console

To add a Google identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Google**.
5. Enter the **Client ID** of the OAuth project you created at Google Cloud Platform. For more information, see Setting up OAuth 2.0 in *Google Cloud Platform Console Help*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.
   - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.
     i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.
     ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.
   a. To apply no principal tags, choose **Inactive**.
   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.
   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Use Google

To enable login with Google in your application, follow the Google documentation for Web.

Successful authentication results in a response object that contains an `id_token` that Amazon Cognito uses to authenticate the user and generate a unique identifier:

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { S3Client } from "@aws-sdk/client-s3";

function signinCallback(authResult) {
  if (authResult['status']['signed_in']) {

     // Attach a Cognito identity pool credentials provider to your service client,
     // passing the Google ID token in the logins map.
     const client = new S3Client({
        region: 'REGION',
        credentials: fromCognitoIdentityPool({
           identityPoolId: 'IDENTITY_POOL_ID',
           logins: {
              'accounts.google.com': authResult['id_token']
           },
           clientConfig: { region: 'REGION' }
        })
     });

     // The provider resolves and caches AWS credentials the first time the
     // client makes a request. Access AWS resources with client here.
  }
}
```

Setting up Sign in with Apple as an identity pool IdP

Amazon Cognito identity pools work with Sign in with Apple to provide federated authentication for your mobile application and web application users. This section explains how to register and set up your application using Sign in with Apple as an identity provider (IdP).

To add Sign in with Apple as an authentication provider to an identity pool, you must complete two procedures. First, integrate Sign in with Apple in an application, and then configure Sign in with Apple in identity pools. For the most up-to-date information about setting up Sign in with Apple, see Configuring Your Environment for Sign in with Apple in the Apple Developer documentation.

Set up Sign in with Apple

To configure Sign in with Apple as an IdP, register your application with the Apple to receive client ID.

1. Create a developer account with Apple.
2. Sign in with your Apple credentials.
3. In the left navigation pane, choose **Certificates, IDs & Profiles**.
4. In the left navigation pane, choose **Identifiers**.
5. On the **Identifiers** page, choose the **+**icon.
6. On the **Register a New Identifier** page, choose **App IDs**, and then choose **Continue**.
7. On the **Register an App ID** page, do the following:

    a. Under **Description**, type a description.

    b. Under **Bundle ID,** type an identifier. Make a note of this **Bundle ID** as you need this value to configure Apple as a provider in the identity pool.

    c. Under **Capabilities**, choose **Sign In with Apple**, and then choose **Edit**.

    d. On the **Sign in with Apple: App ID Configuration** page, select the appropriate setting for your app. Then choose **Save**.

    e. Choose **Continue**.
8. On the **Confirm your App ID** page, choose **Register**.
9. Proceed to step 10 if you want to integrate Sign in with Apple with a native iOS application. Step 11 is for applications that you want to integrate with Sign in with Apple JS.
10. On the **Identifiers** page, choose the **App IDs** menu, then **Services IDs**. Choose the **+** icon.
11. On the **Register a New Identifier** page, choose **Services IDs**, and then choose **Continue**.
12. On the **Register a Services ID** page, do the following:

    a. Under **Description**, type a description.

    b. Under **Identifier**, type an identifier. Make a note of the services ID as you need this value to configure Apple as a provider in your identity pool.

    c. Select **Sign In with Apple** and then choose **Configure**.

    d. On the **Web Authentication Configuration** page, choose a **Primary App ID**. Under **Website URLs**, choose the **+** icon. For **Domains and Subdomains**, enter the domain name of your app. In **Return URLs,** enter the callback URL where the authorization redirects the user after they authenticate through Sign in with Apple.

    e. Choose **Next**.

    f. Choose **Continue**, and then choose **Register**.
13. In the left navigation pane, choose **Keys**.

14. On the **Keys** page, choose the **+** icon.
15. On the **Register a New Key** page, do the following:

    a. Under **Key Name**, type a key name.
    b. Choose **Sign In with Apple**, and then choose **Configure**.
    c. On the **Configure Key** page, choose a **Primary App ID** and then choose **Save**.
    d. Choose **Continue**, and then choose **Register**.

> **Note**
>
> To integrate Sign in with Apple with a native iOS application, see Implementing User Authentication with Sign in with Apple.
> To integrate Sign in with Apple in a platform other than native iOS, see Sign in with Apple JS.

Configure the external provider in the Amazon Cognito federated identities console

Use the following procedure to configure your external provider.

To add a Sign in with Apple identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Sign in with Apple**.
5. Enter the **Services ID** of the OAuth project you created with Apple Developer. For more information, see Authenticating users with Sign in with Apple in *Sign in with Apple Documentation*.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

    - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

        i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that

will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

   a. To apply no principal tags, choose **Inactive**.

   b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

   c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

8. Select **Save changes**.

Sign in with Apple as a provider in the Amazon Cognito federated identities CLI examples

This example creates an identity pool named `MyIdentityPool` with Sign in with Apple as an IdP.

```
aws cognito-identity create-identity-pool --identity-
pool-name MyIdentityPool --supported-login-providers
appleid.apple.com="sameple.apple.clientid"
```

For more information, see Create identity pool

Generate an Amazon Cognito identity ID

This example generates (or retrieves) an Amazon Cognito ID. This is a public API so you don't need any credentials to call this API.

```
aws cognito-identity get-id --identity-pool-id SampleIdentityPoolId --
logins appleid.apple.com="SignInWithAppleIdToken"
```

For more information, see get-id.

Get credentials for an Amazon Cognito identity ID

This example returns credentials for the provided identity ID and Sign in with Apple login. This is a public API so you don't need any credentials to call this API.

```
aws cognito-identity get-credentials-for-identity --identity-id
SampleIdentityId --logins appleid.apple.com="SignInWithAppleIdToken"
```

For more information, see get-credentials-for-identity

Use Sign in with Apple: Android

Apple doesn't provide an SDK that supports Sign in with Apple for Android. You can use the web flow in a web view instead.

- To configure Sign in with Apple in your application, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.
- To add a **Sign in with Apple** button to your Android user interface, follow Displaying Sign in with Apple buttons on the web in the Apple documentation.
- To securely authenticate users with Sign in with Apple, follow Authenticating Users with Sign in with Apple in the Apple documentation.

Sign in with Apple uses a session object to track its state. Amazon Cognito uses the ID token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
@Override
public void onSuccess(Bundle response) {
    String token = response.getString("id_token");
    Map<String, String> logins = new HashMap<String, String>();
    logins.put("appleid.apple.com", token);
    credentialsProvider.setLogins(logins);
}
```

Use Sign in with Apple: iOS - Objective-C

Apple provided SDK support for Sign in with Apple in native iOS applications. To implement user authentication with Sign in with Apple in native iOS devices, follow Implementing User Authentication with Sign in with Apple in the Apple documentation.

Amazon Cognito uses the ID token to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
(void)finishedWithAuth: (ASAuthorizationAppleIDCredential *)auth error: (NSError *)
 error {
```

```
        NSString *idToken = [ASAuthorizationAppleIDCredential
 objectForKey:@"identityToken"];
        credentialsProvider.logins = @{ "appleid.apple.com": idToken };
    }
```

Use Sign in with Apple: iOS - Swift

Apple provided SDK support for Sign in with Apple in native iOS applications. To implement user authentication with Sign in with Apple in native iOS devices, follow Implementing User Authentication with Sign in with Apple in the Apple documentation.

Amazon Cognito uses the ID token to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

For more information about how to set up Sign in with Apple in iOS, see Set up Sign in with Apple

```
func finishedWithAuth(auth: ASAuthorizationAppleIDCredential!, error: NSError!) {
    if error != nil {
      print(error.localizedDescription)
    }
    else {
      let idToken = auth.identityToken,
      credentialsProvider.logins = ["appleid.apple.com": idToken!]
    }
}
```

Use Sign in with Apple: JavaScript

Apple doesn't provide an SDK that supports Sign in with Apple for JavaScript. You can use the web flow in a web view instead.

- To configure Sign in with Apple in your application, follow Configuring Your Web page for Sign In with Apple in the Apple documentation.
- To add a **Sign in with Apple** button to your JavaScript user interface, follow Displaying Sign in with Apple buttons on the web in the Apple documentation.
- To securely authenticate users with Sign in with Apple, follow Authenticating Users with Sign in with Apple in the Apple documentation.

Sign in with Apple uses a session object to track its state. Amazon Cognito uses the ID token from this session object to authenticate the user, generate the unique identifier, and, if needed, grant the user access to other AWS resources.

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { S3Client } from "@aws-sdk/client-s3";

function signinCallback(authResult) {
     // Attach a Cognito identity pool credentials provider to your service client,
     // passing the Apple ID token in the logins map.
     const client = new S3Client({
        region: 'REGION',
        credentials: fromCognitoIdentityPool({
           identityPoolId: 'IDENTITY_POOL_ID',
           logins: {
              'appleid.apple.com': authResult['id_token']
           },
           clientConfig: { region: 'REGION' }
        })
     });

     // The provider resolves and caches AWS credentials the first time the
     // client makes a request. Access AWS resources with client here.
}
```

Setting up an OIDC provider as an identity pool IdP

OpenID Connect is an open standard for authentication that a number of login providers support. With Amazon Cognito, you can link identities with OpenID Connect providers that you configure through AWS Identity and Access Management.

Adding an OpenID Connect provider

For information about how to create an OpenID Connect provider, see Creating OpenID Connect (OIDC) identity providers in the *AWS Identity and Access Management User Guide*.

Associating a provider with Amazon Cognito

To add an OIDC identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.

3. Select **Add identity provider**.
4. Choose **OpenID Connect (OIDC)**.
5. Choose an **OIDC identity provider** from the IAM IdPs in your AWS account. If you want to add a new SAML provider, choose **Create new provider** to navigate to the IAM console.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

    - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

        i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

        ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

    a. To apply no principal tags, choose **Inactive**.

    b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

    c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

You can associate multiple OpenID Connect providers with a single identity pool.

Using OpenID Connect

Refer to your provider's documentation for how to sign in and receive an ID token.

After you have a token, add the token to the logins map. Use the URI of your provider as the key.

Validating an OpenID Connect token

When you first integrate with Amazon Cognito, you might receive an `InvalidToken` exception. It is important to understand how Amazon Cognito validates OpenID Connect (OIDC) tokens.

> **Note**
>
> As specified here (https://tools.ietf.org/html/rfc7523), Amazon Cognito provides a grace period of 5 minutes to handle any clock skew between systems.

1. The `iss` parameter must match the key that the logins map uses (such as login.provider.com).
2. The signature must be valid. The signature must be verifiable via an RSA public key.

   > **Note**
   >
   > Identity pools maintain a cache of the OIDC IdP signing key for a brief period. If your provider changes their signing key, Amazon Cognito might return a `NoKeyFound` error until this cache refreshes. If you encounter this error, wait about ten minutes for your identity pool to refresh the signing key.

3. The fingerprint of the certificate public key matches the fingerprint that you set in IAM when you created your OIDC provider.
4. If the `azp` parameter is present, check this value against listed client IDs in your OIDC provider.
5. If the `azp` parameter isn't present, check the `aud` parameter against listed client IDs in your OIDC provider.

The website jwt.io is a valuable resource that you can use to decode tokens and verify these values.

Android

```
Map<String, String> logins = new HashMap<String, String>();
logins.put("login.provider.com", token);
credentialsProvider.setLogins(logins);
```

iOS - Objective-C

```
credentialsProvider.logins = @{ "login.provider.com": token }
```

JavaScript

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";
import { S3Client } from "@aws-sdk/client-s3";

const client = new S3Client({
 region: 'REGION',
 credentials: fromCognitoIdentityPool({
    identityPoolId: 'IDENTITY_POOL_ID',
    logins: {
       'login.provider.com': token
    },
    clientConfig: { region: 'REGION' }
 })
});
```

Setting up a SAML provider as an identity pool IdP

With Amazon Cognito identity pools, you can authenticate users with identity providers (IdPs) through SAML 2.0. You can use an IdP that supports SAML with Amazon Cognito to provide a simple onboarding flow for your users. Your SAML-supporting IdP specifies the IAM roles that your users can assume. This way, different users can receive different sets of permissions.

Configuring your identity pool for a SAML IdP

The following steps describe how to configure your identity pool to use a SAML-based IdP.

> **Note**
>
> Before you configure your identity pool to support a SAML provider, first configure the SAML IdP in the IAM console. For more information, see Integrating third-party SAML solution providers with AWS in the *IAM User Guide*.

To add a SAML identity provider (IdP)

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **SAML**.

5. Choose a **SAML identity provider** from the IAM IdPs in your AWS account. If you want to add a new SAML provider, choose **Create new provider** to navigate to the IAM console.
6. To set the role that Amazon Cognito requests when it issues credentials to users who have authenticated with this provider, configure **Role settings**.

    - You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**.

        i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

        ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.
7. To change the principal tags that Amazon Cognito assigns when it issues credentials to users who have authenticated with this provider, configure **Attributes for access control**.

    a. To apply no principal tags, choose **Inactive**.

    b. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

    c. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.
8. Select **Save changes**.

Configuring your SAML IdP

After you create the SAML provider, configure your SAML IdP to add relying party trust between your IdP and AWS. With many IdPs, you can specify a URL that the IdP can use to read relying party information and certificates from an XML document. For AWS, you can use https://signin.aws.amazon.com/static/saml-metadata.xml. The next step is to configure the SAML assertion response from your IdP to populate the claims that AWS needs. For details on the claim configuration, see Configuring SAML assertions for authentication response.

When your SAML IdP includes more than one signing certificate in SAML metadata, at sign-in your identity pool determines that the SAML assertion is valid if it matches any certificate in the SAML metadata.

Customizing your user role with SAML

When you use SAML with Amazon Cognito Identity, you can customize the role for the end user. Amazon Cognito only supports the enhanced flow with the SAML-based IdP. You don't need to specify an authenticated or unauthenticated role for the identity pool to use a SAML-based IdP. The `https://aws.amazon.com/SAML/Attributes/Role` claim attribute specifies one or more pairs of comma -delimited role and provider ARN. These are the roles that the user can assume. You can configure the SAML IdP to populate the role attributes based on the user attribute information available from the IdP. If you receive multiple roles in the SAML assertion, populate the optional `customRoleArn` parameter when you call `getCredentialsForIdentity`. The user assumes this `customRoleArn` if the role matches one in the claim in the SAML assertion.

Authenticating users with a SAML IdP

To federate with the SAML-based IdP, determine the URL where the user initiates the login. AWS federation uses IdP-initiated login. In AD FS 2.0, the URL takes the form of `https://<fqdn>/adfs/ls/IdpInitiatedSignOn.aspx?loginToRp=urn:amazon:webservices`.

To add support for your SAML IdP in Amazon Cognito, first authenticate users with your SAML identity provider from your iOS or Android application. The code that you use to integrate and authenticate with the SAML IdP is specific to SAML providers. After you authenticate your user, you can use Amazon Cognito APIs to provide the resulting SAML assertion to Amazon Cognito Identity .

You can't repeat, or *replay*, a SAML assertion in the `Logins` map of your identity pool API request. A replayed SAML assertion has an assertion ID that duplicates the ID of an earlier API request. API operations that can accept a SAML assertion in the `Logins` map include GetId, GetCredentialsForIdentity, GetOpenIdToken, and GetOpenIDTokenForDeveloperIdentity. You can replay a SAML assertion ID one time per API request in an identity pool authentication flow. For example, you can supply the same SAML assertion in a `GetId` request and a subsequent `GetCredentialsForIdentity` request, but not in a second `GetId` request.

Developer-authenticated identities

Amazon Cognito supports developer-authenticated identities, in addition to web identity federation through Setting up Facebook as an identity pools IdP, Setting up Google as an identity pool IdP, Setting up Login with Amazon as an identity pools IdP, and Setting up Sign in with Apple as an identity pool IdP. With developer-authenticated identities, you can register and authenticate users through your own existing authentication process, while still using Amazon

Cognito to synchronize user data and access AWS resources. Using developer-authenticated identities involves interaction between the end user device, your backend for authentication, and Amazon Cognito. For more details, see Understanding Amazon Cognito Authentication Part 2: Developer Authenticated Identities in the AWS blog.

Understanding the authentication flow

The GetOpenIdTokenForDeveloperIdentity API operation can initiate developer authentication for both enhanced and basic authentication. This API authenticates a request with administrative credentials. The `Logins` map is an identity pool developer provider name like `login.mydevprovider` paired with a custom identifier.

Example:

```
"Logins": {
        "login.mydevprovider": "my developer identifier"
    }
```

Enhanced authentication

Call the GetCredentialsForIdentity API operation with a `Logins` map where the key is the issuer of the token returned by `GetOpenIdTokenForDeveloperIdentity`, and the value is the token itself.

The issuer (and therefore the required logins key) depends on the Region of your identity pool. See Token issuer by Region for the complete list of Regions and their token issuers.

Example for an identity pool in a Region that uses the global issuer:

```
"Logins": {
        "cognito-identity.amazonaws.com": "eyJra12345EXAMPLE"
    }
```

Example for an identity pool in a newer Region (such as eu-south-2):

```
"Logins": {
        "cognito-identity.eu-south-2.amazonaws.com": "eyJra12345EXAMPLE"
    }
```

Important

The logins map key must exactly match the `iss` (issuer) claim in the token. If they do not match, `GetCredentialsForIdentity` returns: `NotAuthorizedException: Invalid login token. Issuer doesn't match providerName`

Tip

To determine the correct logins key programmatically, decode the token (a JWT) and read the `iss` claim value. Use that value as the logins map key.

`GetCredentialsForIdentity` with developer-authenticated identities returns temporary credentials for the default authenticated role of the identity pool.

Basic authentication

Call the AssumeRoleWithWebIdentity API operation and request the `RoleArn` of any IAM role that has an appropriate trust relationship defined. Set the value of `WebIdentityToken` to the token obtained from `GetOpenIdTokenForDeveloperIdentity`.

For information on the developer-authenticated identities authflow and how they differ from external-provider identities, see Identity pools authentication flow.

Define a developer provider name and associate it with an identity pool

To use developer-authenticated identities, you'll need an identity pool associated with your developer provider. To do so, follow these steps:

To add a custom developer provider

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **User access** tab.
3. Select **Add identity provider**.
4. Choose **Custom developer provider**.
5. Enter a **Developer provider name**. You can't change or delete your developer provider after you add it.

6. Select **Save changes**.

Note: Once the provider name has been set, it cannot be changed.

Implement an identity provider

Android

To use developer-authenticated identities, implement your own identity provider class that extends `AWSAbstractCognitoIdentityProvider`. Your identity provider class should return a response object containing the token as an attribute.

Following is a basic example of an identity provider.

```
public class DeveloperAuthenticationProvider extends
 AWSAbstractCognitoDeveloperIdentityProvider {

  private static final String developerProvider = "<Developer_provider_name>";

  public DeveloperAuthenticationProvider(String accountId, String identityPoolId,
 Regions region) {
    super(accountId, identityPoolId, region);
    // Initialize any other objects needed here.
  }

  // Return the developer provider name which you choose while setting up the
  // identity pool in the &COG; Console

  @Override
  public String getProviderName() {
    return developerProvider;
  }

  // Use the refresh method to communicate with your backend to get an
  // identityId and token.

  @Override
  public String refresh() {

    // Override the existing token
    setToken(null);
```

```
    // Get the identityId and token by making a call to your backend
    // (Call to your backend)

    // Call the update method with updated identityId and token to make sure
    // these are ready to be used from Credentials Provider.

    update(identityId, token);
    return token;

  }

  // If the app has a valid identityId return it, otherwise get a valid
  // identityId from your backend.

  @Override
  public String getIdentityId() {

    // Load the identityId from the cache
    identityId = cachedIdentityId;

    if (identityId == null) {
       // Call to your backend
    } else {
       return identityId;
    }

  }
}
```

To use this identity provider, you have to pass it into `CognitoCachingCredentialsProvider`. Here's an example:

```
DeveloperAuthenticationProvider developerProvider = new
 DeveloperAuthenticationProvider( null, "IDENTITYPOOLID", Regions.USEAST1);
CognitoCachingCredentialsProvider credentialsProvider = new
 CognitoCachingCredentialsProvider( context, developerProvider, Regions.USEAST1);
```

Note

Older versions of the AWS SDK for Android (2.x and earlier) use `cognito-identity.amazonaws.com` as the internal provider name for developer-authenticated

identities. If your identity pool is in a newer Region that uses a regionalized issuer, upgrade to a current SDK version. For more information, see Regional token issuer format.

iOS - objective-C

To use developer-authenticated identities, implement your own identity provider class that extends AWSCognitoCredentialsProviderHelper. Your identity provider class should return a response object containing the token as an attribute.

```
@implementation DeveloperAuthenticatedIdentityProvider
/*
 * Use the token method to communicate with your backend to get an
 * identityId and token.
 */

- (AWSTask <NSString*> *) token {
    //Write code to call your backend:
    //Pass username/password to backend or some sort of token to authenticate user
    //If successful, from backend call getOpenIdTokenForDeveloperIdentity with logins
 map
    //containing "your.provider.name":"enduser.username"
    //Return the identity id and token to client
    //You can use AWSTaskCompletionSource to do this asynchronously

    // Set the identity id and return the token
    self.identityId = response.identityId;
    return [AWSTask taskWithResult:response.token];
}

@end
```

To use this identity provider, pass it into `AWSCognitoCredentialsProvider` as shown in the following example:

```
DeveloperAuthenticatedIdentityProvider * devAuth =
 [[DeveloperAuthenticatedIdentityProvider alloc]
 initWithRegionType:AWSRegionYOUR_IDENTITY_POOL_REGION
                                        identityPoolId:@"YOUR_IDENTITY_POOL_ID"
                                       useEnhancedFlow:YES
                                identityProviderManager:nil];
```

```
AWSCognitoCredentialsProvider *credentialsProvider = [[AWSCognitoCredentialsProvider
 alloc]

 initWithRegionType:AWSRegionYOUR_IDENTITY_POOL_REGION
                                                      identityProvider:devAuth];
```

If you want to support both unauthenticated identities and developer-authenticated identities, override the `logins` method in your `AWSCognitoCredentialsProviderHelper` implementation.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else{
        return [super logins];
    }
}
```

If you want to support developer-authenticated identities and social providers, you must manage who the current provider is in your `logins` implementation of `AWSCognitoCredentialsProviderHelper`.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else if (/*logic to determine if user is Facebook*/){
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook :
 [FBSDKAccessToken currentAccessToken] }];
    }else {
        return [super logins];
    }
}
```

Note

Older versions of the AWS SDK for iOS (2.x and earlier) use `cognito-identity.amazonaws.com` as the internal provider name for developer-authenticated identities. If your identity pool is in a newer Region that uses a regionalized issuer, upgrade to a current SDK version. For more information, see Regional token issuer format.

iOS - swift

To use developer-authenticated identities, implement your own identity provider class that extends AWSCognitoCredentialsProviderHelper. Your identity provider class should return a response object containing the token as an attribute.

```
import AWSCore
/*
 * Use the token method to communicate with your backend to get an
 * identityId and token.
 */
class DeveloperAuthenticatedIdentityProvider : AWSCognitoCredentialsProviderHelper {
    override func token() -> AWSTask<NSString> {
    //Write code to call your backend:
    //pass username/password to backend or some sort of token to authenticate user, if
 successful,
    //from backend call getOpenIdTokenForDeveloperIdentity with logins map containing
 "your.provider.name":"enduser.username"
    //return the identity id and token to client
    //You can use AWSTaskCompletionSource to do this asynchronously

    // Set the identity id and return the token
    self.identityId = resultFromAbove.identityId
    return AWSTask(result: resultFromAbove.token)
}
```

To use this identity provider, pass it into `AWSCognitoCredentialsProvider` as shown in the following example:

```
let devAuth =
 DeveloperAuthenticatedIdentityProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
 identityPoolId: "YOUR_IDENTITY_POOL_ID", useEnhancedFlow: true,
 identityProviderManager:nil)
let credentialsProvider =
 AWSCognitoCredentialsProvider(regionType: .YOUR_IDENTITY_POOL_REGION,
 identityProvider:devAuth)
let configuration = AWSServiceConfiguration(region: .YOUR_IDENTITY_POOL_REGION,
 credentialsProvider:credentialsProvider)
AWSServiceManager.default().defaultServiceConfiguration = configuration
```

If you want to support both unauthenticated identities and developer-authenticated identities, override the `logins` method in your `AWSCognitoCredentialsProviderHelper` implementation.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else {
        return super.logins()
    }
}
```

If you want to support developer-authenticated identities and social providers, you must manage who the current provider is in your `logins` implementation of `AWSCognitoCredentialsProviderHelper`.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else if (/*logic to determine if user is Facebook*/){
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }else {
        return super.logins()
    }
}
```

JavaScript

Once you obtain an identity ID and session token from your backend, you will pass them to the `fromCognitoIdentity` provider from `@aws-sdk/credential-providers`. The logins key must match the token's `iss` claim. For identity pools in Regions that use the global issuer, use `cognito-identity.amazonaws.com`. For identity pools in other Regions, use `cognito-identity.`*`region`*`.amazonaws.com`. For more information, see Regional token issuer format.

```
import { fromCognitoIdentity } from "@aws-sdk/credential-providers";

const token = 'TOKEN_RETURNED_FROM_YOUR_PROVIDER';
```

```
const region = 'IDENTITY_POOL_REGION';

// Extract the issuer from the token
const payload = token.split('.')[1];
const claims = JSON.parse(Buffer.from(payload, 'base64url').toString());
const providerName = claims.iss;

// Your backend already established the identity, so use fromCognitoIdentity
// (which calls GetCredentialsForIdentity directly for the identity ID you pass)
// rather than fromCognitoIdentityPool (which would call GetId and resolve a new
 identity).
const credentials = fromCognitoIdentity({
   identityId: 'IDENTITY_ID_RETURNED_FROM_YOUR_PROVIDER',
   logins: {
      [providerName]: token
   },
   clientConfig: { region }
});

// Attach the provider to each service client that needs credentials.
const client = new SomeServiceClient({ region, credentials });
```

Updating the logins map (Android and iOS only)

Android

After successfully authenticating the user with your authentication system, update the logins map with the developer provider name and a developer user identifier. This is an alphanumeric string that uniquely identifies a user in your authentication system. Be sure to call the `refresh` method after updating the logins map as the `identityId` might have changed:

```
HashMap<String, String> loginsMap = new HashMap<String, String>();
loginsMap.put(developerAuthenticationProvider.getProviderName(),
 developerUserIdentifier);

credentialsProvider.setLogins(loginsMap);
credentialsProvider.refresh();
```

iOS - objective-C

The iOS SDK only calls your `logins` method to get the latest logins map if there are no credentials or they have expired. If you want to force the SDK to obtain new credentials (for example, your

end user went from unauthenticated to authenticated and you want credentials against the authenticated user), call `clearCredentials` on your `credentialsProvider`.

```
[credentialsProvider clearCredentials];
```

iOS - swift

The iOS SDK only calls your `logins` method to get the latest logins map if there are no credentials or they have expired. If you want to force the SDK to obtain new credentials (e.g., your end user went from unauthenticated to authenticated and you want credentials against the authenticated user), call `clearCredentials` on your `credentialsProvider`.

```
credentialsProvider.clearCredentials()
```

Getting a token (server side)

You obtain a token by calling GetOpenIdTokenForDeveloperIdentity. This API must be invoked from your backend using AWS developer credentials. It must not be invoked from the client SDK. The API receives the Cognito identity pool ID; a logins map containing your identity provider name as the key and identifier as the value; and optionally a Cognito identity ID (for example, you are making an unauthenticated user authenticated). The identifier can be the username of your user, an email address, or a numerical value. The API responds to your call with a unique Cognito ID for your user and an OpenID Connect token for the end user.

A few things to keep in mind about the token returned by `GetOpenIdTokenForDeveloperIdentity`:

- You can specify a custom expiration time for the token so you can cache it. If you don't provide any custom expiration time, the token is valid for 15 minutes.
- The maximum token duration that you can set is 24 hours.
- Be mindful of the security implications of increasing the token duration. If an attacker obtains this token, they can exchange it for AWS credentials for the end user for the token duration.

The following Java snippet shows how to initialize an Amazon Cognito client and retrieve a token for a developer-authenticated identity.

```
// authenticate your end user as appropriate
```

```
// ....

// if authenticated, initialize a cognito client with your AWS developer credentials
AmazonCognitoIdentity identityClient = new AmazonCognitoIdentityClient(
  new BasicAWSCredentials("access_key_id", "secret_access_key")
);

// create a new request to retrieve the token for your end user
GetOpenIdTokenForDeveloperIdentityRequest request =
  new GetOpenIdTokenForDeveloperIdentityRequest();
request.setIdentityPoolId("YOUR_COGNITO_IDENTITY_POOL_ID");

request.setIdentityId("YOUR_COGNITO_IDENTITY_ID"); //optional, set this if your client
 has an
                                                    //identity ID that you want to link
 to this
                                                    //developer account

// set up your logins map with the username of your end user
HashMap<String,String> logins = new HashMap<>();
logins.put("YOUR_IDENTITY_PROVIDER_NAME","YOUR_END_USER_IDENTIFIER");
request.setLogins(logins);

// optionally set token duration (in seconds)
request.setTokenDuration(60 * 15l);
GetOpenIdTokenForDeveloperIdentityResult response =
  identityClient.getOpenIdTokenForDeveloperIdentity(request);

// obtain identity id and token to return to your client
String identityId = response.getIdentityId();
String token = response.getToken();

//code to return identity id and token to client
//...
```

Following the preceding steps, you should be able to integrate developer-authenticated identities in your app. If you have any issues or questions please feel free to post in our forums.

Regional token issuer format

The OpenID Connect token returned by `GetOpenIdTokenForDeveloperIdentity` contains an `iss` (issuer) claim. The format of this claim depends on the AWS Region of the identity pool.

When you call `GetCredentialsForIdentity` (enhanced flow), the logins map key **must exactly match** the token's `iss` claim.

Token issuer by Region

The token issuer value depends on the Region of your identity pool.

- **Regions that use the global issuer** (us-east-1, us-east-2, us-west-1, us-west-2, ap-northeast-1, ap-northeast-2, ap-northeast-3, ap-south-1, ap-southeast-1, ap-southeast-2, ca-central-1, eu-central-1, eu-north-1, eu-west-1, eu-west-2, eu-west-3, sa-east-1): `cognito-identity.amazonaws.com`
- **The China Regions** (cn-north-1): `cognito-identity.cn-north-1.amazonaws.com.cn`
- **All remaining commercial AWS Regions**: `cognito-identity.`*`region`*`.amazonaws.com`

Determining the logins key programmatically

Rather than hardcoding the logins key, you can extract it from the token:

Java

```
import java.util.Base64;
import org.json.JSONObject;

String[] parts = token.split("\\.");
String payload = new String(Base64.getUrlDecoder().decode(parts[1]));
String issuer = new JSONObject(payload).getString("iss");
// Use 'issuer' as the logins map key in GetCredentialsForIdentity
```

Python

```
import base64, json

payload = token.split('.')[1]
# Add padding if needed
payload += '=' * (4 - len(payload) % 4)
claims = json.loads(base64.urlsafe_b64decode(payload))
issuer = claims['iss']
# Use 'issuer' as the logins map key
```

JavaScript

```
const payload = token.split('.')[1];
const claims = JSON.parse(Buffer.from(payload, 'base64url').toString());
const issuer = claims.iss;
// Use 'issuer' as the logins map key
```

Migration considerations

If you are migrating an identity pool to a Region that uses a regionalized issuer, be aware of the following:

1. Clients that hardcode `cognito-identity.amazonaws.com` as the logins key will fail against the new Region's identity pool.
2. Client code must be updated to use the regionalized issuer (`cognito-identity.`*`new-region`*`.amazonaws.com`), or to dynamically read the `iss` claim from the token (recommended).
3. Older versions of the AWS Mobile SDKs hardcode the global issuer. This includes the AWS SDK for Android 2.x and earlier, and the AWS SDK for iOS 2.x and earlier. If your identity pool uses a regionalized issuer, upgrade to a current SDK version.

Connect to an existing social identity

All linking of providers when you are using developer-authenticated identities must be done from your backend. To connect a custom identity to a user's social identity (Login with Amazon, Sign in with Apple, Facebook, or Google), add the identity provider token to the logins map when you call GetOpenIdTokenForDeveloperIdentity. To make this possible, when you call your backend from your client SDK to authenticate your end user, additionally pass the end user's social provider token.

For example, if you are trying to link a custom identity to Facebook, you would add the Facebook token in addition to your identity provider identifier to the logins map when you call `GetOpenIdTokenForDeveloperIdentity`.

```
logins.put("YOUR_IDENTITY_PROVIDER_NAME","YOUR_END_USER_IDENTIFIER");
logins.put("graph.facebook.com","END_USERS_FACEBOOK_ACCESSTOKEN");
```

Supporting transition between providers

Android

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Sign in with Apple, Facebook, or Google) along with developer-authenticated identities. The essential difference between developer-authenticated identities and other identities (unauthenticated identities and authenticated identities using public provider) is the way the identityId and token are obtained. For other identities, the mobile application will interact directly with Amazon Cognito instead of contacting your authentication system. So the mobile application should be able to support two distinct flows depending on the choice made by the app user. For this, you will have to make some changes to the custom identity provider.

The `refresh` method checks the logins map. If the map is not empty and has a key with developer provider name, then call your backend. Otherwise, call the getIdentityId method and return null.

```
public String refresh() {

   setToken(null);

   // If the logins map is not empty make a call to your backend
   // to get the token and identityId
   if (getProviderName() != null &&
      !this.loginsMap.isEmpty() &&
      this.loginsMap.containsKey(getProviderName())) {

      /**
       * This is where you would call your backend
       **/

      // now set the returned identity id and token in the provider
      update(identityId, token);
      return token;

   } else {
      // Call getIdentityId method and return null
      this.getIdentityId();
      return null;
   }
}
```

Similarly the `getIdentityId` method will have two flows depending on the contents of the logins map:

```
public String getIdentityId() {

   // Load the identityId from the cache
   identityId = cachedIdentityId;

   if (identityId == null) {

      // If the logins map is not empty make a call to your backend
      // to get the token and identityId

      if (getProviderName() != null && !this.loginsMap.isEmpty()
         && this.loginsMap.containsKey(getProviderName())) {

         /**
           * This is where you would call your backend
          **/

         // now set the returned identity id and token in the provider
         update(identityId, token);
         return token;

      } else {
         // Otherwise call &COG; using getIdentityId of super class
         return super.getIdentityId();
      }

   } else {
      return identityId;
   }

}
```

iOS - objective-C

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Sign in with Apple, Facebook, or Google) along with developer-authenticated identities. To do this, override the AWSCognitoCredentialsProviderHelper `logins` method to be able to return the correct logins map based on the current identity provider.

This example shows you how you might pivot between unauthenticated, Facebook and developer-authenticated.

```
- (AWSTask<NSDictionary<NSString *, NSString *> *> *)logins {
    if(/*logic to determine if user is unauthenticated*/) {
        return [AWSTask taskWithResult:nil];
    }else if (/*logic to determine if user is Facebook*/){
        return [AWSTask taskWithResult: @{ AWSIdentityProviderFacebook :
 [FBSDKAccessToken currentAccessToken] }];
    }else {
        return [super logins];
    }
}
```

When you transition from unauthenticated to authenticated, you should call `[credentialsProvider clearCredentials];` to force the SDK to get new authenticated credentials. When you switch between two authenticated providers and you aren't trying to link the two providers (for example, you are not providing tokens for multiple providers in your logins dictionary), call `[credentialsProvider clearKeychain];`. This will clear both the credentials and identity and force the SDK to get new ones.

iOS - swift

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Sign in with Apple, Facebook, or Google) along with developer-authenticated identities. To do this, override the AWSCognitoCredentialsProviderHelper `logins` method to be able to return the correct logins map based on the current identity provider. This example shows you how you might pivot between unauthenticated, Facebook and developer-authenticated.

```
override func logins () -> AWSTask<NSDictionary> {
    if(/*logic to determine if user is unauthenticated*/) {
        return AWSTask(result:nil)
    }else if (/*logic to determine if user is Facebook*/){
        if let token = AccessToken.current?.authenticationToken {
            return AWSTask(result: [AWSIdentityProviderFacebook:token])
        }
        return AWSTask(error:NSError(domain: "Facebook Login", code: -1 , userInfo:
 ["Facebook" : "No current Facebook access token"]))
    }else {
        return super.logins()
```

```
    }
}
```

When you transition from unauthenticated to authenticated, you should call `credentialsProvider.clearCredentials()` to force the SDK to get new authenticated credentials. When you switch between two authenticated providers and you aren't trying to link the two providers (i.e. you are not providing tokens for multiple providers in your logins dictionary), you should call `credentialsProvider.clearKeychain()`. This will clear both the credentials and identity and force the SDK to get new ones.

Unity

Your application might require supporting unauthenticated identities or authenticated identities using public providers (Login with Amazon, Sign in with Apple, Facebook, or Google) along with developer-authenticated identities. The essential difference between developer-authenticated identities and other identities (unauthenticated identities and authenticated identities using public provider) is the way the identityId and token are obtained. For other identities, the mobile application will interact directly with Amazon Cognito instead of contacting your authentication system. The mobile application should be able to support two distinct flows depending on the choice made by the app user. For this you will have to make some changes to the custom identity provider.

The recommended way to do it in Unity is to extend your identity provider from AmazonCognitoEnhancedIdentityProvide instead of AbstractCognitoIdentityProvider, and call the parent RefreshAsync method instead of your own in case the user is not authenticated with your own backend. If the user is authenticated, you can use the same flow explained before.

Switching unauthenticated users to authenticated users

Amazon Cognito identity pools support both authenticated and unauthenticated users. Unauthenticated users receive access to your AWS resources even if they aren't logged in with any of your identity providers (IdPs). This degree of access is useful to display content to users before they log in. Each unauthenticated user has a unique identity in the identity pool, even though they haven't been individually logged in and authenticated.

This section describes the case where your user chooses to switch from logging in with an unauthenticated identity to using an authenticated identity.

Android

Users can log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is merged automatically.

Your application is informed of a profile merge through the `IdentityChangedListener` interface. Implement the `identityChanged` method in the interface to receive these messages:

```
@override
public void identityChanged(String oldIdentityId, String newIdentityId) {
    // handle the change
}
```

iOS - objective-C

Users can log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is merged automatically.

`NSNotificationCenter` informs your application of a profile merge:

```
[[NSNotificationCenter defaultCenter] addObserver:self
                                      selector:@selector(identityIdDidChange:)
                                      name:AWSCognitoIdentityIdChangedNotification
                                      object:nil];

-(void)identityDidChange:(NSNotification*)notification {
    NSDictionary *userInfo = notification.userInfo;
    NSLog(@"identity changed from %@ to %@",
        [userInfo objectForKey:AWSCognitoNotificationPreviousId],
        [userInfo objectForKey:AWSCognitoNotificationNewId]);
}
```

iOS - swift

Users can log in to your application as unauthenticated guests. Eventually they might decide to log in using one of the supported IdPs. Amazon Cognito makes sure that an old identity retains the same unique identifier as the new one, and that the profile data is merged automatically.

`NSNotificationCenter` informs your application of a profile merge:

```
[NSNotificationCenter.defaultCenter().addObserver(observer: self
   selector:"identityDidChange"
   name:AWSCognitoIdentityIdChangedNotification
   object:nil)

func identityDidChange(notification: NSNotification!) {
  if let userInfo = notification.userInfo as? [String: AnyObject] {
    print("identity changed from: \(userInfo[AWSCognitoNotificationPreviousId])
    to: \(userInfo[AWSCognitoNotificationNewId])")
  }
}
```

JavaScript

Initially unauthenticated user

Users typically start with the unauthenticated role. For this role, you create a `fromCognitoIdentityPool` credentials provider without a `logins` property and attach it to your service clients. In this case, your default configuration might look like the following:

```
import { fromCognitoIdentityPool } from "@aws-sdk/credential-providers";

// Create a credentials provider without a logins map for the unauthenticated role.
const region = "us-east-1";
const credentials = fromCognitoIdentityPool({
    identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
    clientConfig: { region }
});

// Attach the provider to each service client that needs credentials.
const client = new SomeServiceClient({ region, credentials });
```

Switch to authenticated user

When an unauthenticated user logs in to an IdP and you have a token, you can switch the user from unauthenticated to authenticated by calling a custom function that creates a new `fromCognitoIdentityPool` provider with the `logins` map that contains the token:

```
// Called when an identity provider has a token for a logged in user
function userLoggedIn(providerName, token) {
```

```
    // Re-create the provider with a logins map to switch to the authenticated role.
    const credentials = fromCognitoIdentityPool({
        identityPoolId: "us-east-1:1699ebc0-7900-4099-b910-2df94f52a030",
        logins: {
            [providerName]: token
        },
        clientConfig: { region }
    });

    // Re-create any service clients so they pick up the new credentials.
    // SomeServiceClient is a placeholder: import the client for the service you're
 calling
    // (for example, S3Client from @aws-sdk/client-s3).
    const client = new SomeServiceClient({ region, credentials });
}
```

Because the `fromCognitoIdentityPool` provider is attached directly to each service client, switching from the unauthenticated to the authenticated role means creating a new provider with the `logins` map and passing it to any service clients that you create afterward. Unlike the AWS SDK for JavaScript v2, the v3 SDK doesn't use a global configuration object, so there are no shared credentials to reset.

For more information about the `fromCognitoIdentityPool` provider, see AWS SDK for JavaScript v3 credential providers.

Getting started with Amazon Cognito identity pools

With Amazon Cognito identity pools, you can create unique identities and assign permissions for users. Your identity pool can bring in identities from the following types of authentication services:

- Users in an Amazon Cognito user pool
- Users who authenticate with external identity providers such as Facebook, Google, Apple, or an OIDC or SAML identity provider.
- Users authenticated via your own existing authentication process

After users authenticate with their provider and present authorization to an identity pool, they get temporary AWS credentials. Users' credentials have permissions that you define for access to other AWS services.

Topics

- Create an identity pool in Amazon Cognito
- Set up an SDK
- Integrate the identity providers
- Get credentials
- Example application for identity pools

Create an identity pool in Amazon Cognito

You can create an identity pool through the Amazon Cognito console, or you can use the AWS Command Line Interface (CLI) or the Amazon Cognito APIs. The following procedure is a general guide to create a new identity pool in the console. You can also skip straight to the console and follow the guided experience and inline help content.

To create a new identity pool in the console

1. Sign in to the Amazon Cognito console and select **Identity pools**. To assign permissions to your IAM principal so that they can create and manage Amazon Cognito resources, refer to AWS managed policies for Amazon Cognito. The `AmazonCognitoPowerUser` policy is sufficient for the creation of identity pools.
2. Choose **Create identity pool**.
3. In **Configure identity pool trust**, choose to set up your identity pool for **Authenticated access**, **Guest access**, or both.
    - If you chose **Authenticated access**, select one or more **Identity types** that you want to set as the source of authenticated identities in your identity pool. If you configure a **Custom developer provider**, you can't modify or delete it after you create your identity pool.
4. In **Configure permissions**, choose a default IAM role for authenticated or guest users in your identity pool.
    a. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
    b. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-`

`identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see Role trust and permissions.

5. In **Connect identity providers**, enter the details of the identity providers (IdPs) that you chose in **Configure identity pool trust**. You might be asked to provide OAuth app client information, choose an Amazon Cognito user pool, choose an IAM IdP, or enter a custom identifier for a developer provider.

   a. Choose the **Role settings** for each IdP. You can assign users from that IdP the **Default role** that you set up when you configured your **Authenticated role**, or you can **Choose role with rules**. With an Amazon Cognito user pool IdP, you can also **Choose role with preferred_role in tokens**. For more information about the `cognito:preferred_role` claim, see Assigning precedence values to groups.

      i. If you chose **Choose role with rules**, enter the source **Claim** from your user's authentication, the **Operator** that you want to compare the claim by, the **Value** that will cause a match to this role choice, and the **Role** that you want to assign when the **Role assignment** matches. Select **Add another** to create an additional rule based on a different condition.

      ii. Choose a **Role resolution**. When your user's claims don't match your rules, you can deny credentials or issue credentials for your **Authenticated role**.

   b. Configure **Attributes for access control** for each IdP. Attributes for access control maps user claims to principal tags that Amazon Cognito applies to their temporary session. You can build IAM policies to filter user access based on the tags that you apply to their session.

      i. To apply no principal tags, choose **Inactive**.

      ii. To apply principal tags based on `sub` and `aud` claims, choose **Use default mappings**.

      iii. To create your own custom schema of attributes to principal tags, choose **Use custom mappings**. Then enter a **Tag key** that you want to source from each **Claim** that you want to represent in a tag.

6. In **Configure properties**, enter a **Name** under **Identity pool name**.

7. Under **Basic (classic) authentication**, choose whether you want to **Activate basic flow**. With the basic flow active, you can bypass the role selections you made for your IdPs and call

AssumeRoleWithWebIdentity directly. For more information, see Identity pools authentication flow.

8. Under **Tags**, choose **Add tag** if you want to apply tags to your identity pool.
9. In **Review and create**, confirm the selections that you made for your new identity pool. Select **Edit** to return to the wizard and change any settings. When you're done, select **Create identity pool**.

Set up an SDK

To use Amazon Cognito identity pools, set up AWS Amplify, the AWS SDK for Java, or the SDK for .NET. For more information, see the following topics.

- Setting up the SDK for JavaScript in the *AWS SDK for JavaScript Developer Guide*
- Amplify Documentation in the *Amplify Dev Center*
- Amazon Cognito credentials provider in the *SDK for .NET Developer Guide*

Integrate the identity providers

Amazon Cognito identity pools (federated identities) support user authentication through Amazon Cognito user pools, federated identity providers—including Amazon, Facebook, Google, Apple, and SAML identity providers—and unauthenticated identities. This feature also supports Developer-authenticated identities, which lets you register and authenticate users via your own backend authentication process.

To learn more about using an Amazon Cognito user pool to create your own user directory, see Authenticating users with Amazon Cognito user pools and Accessing AWS services using an identity pool after sign-in.

To learn more about using external identity providers, see Identity pools third-party identity providers.

To learn more about integrating your own backend authentication process, see Developer-authenticated identities.

Get credentials

Amazon Cognito identity pools provide temporary AWS credentials for users who are guests (unauthenticated) and for users who have authenticated and received a token. With those AWS

credentials, your app can securely access a backend in AWS or outside AWS through Amazon API Gateway. See Getting credentials.

Example application for identity pools

The most common use case for Amazon Cognito identity pools is to federate users from multiple sign-in systems and deliver temporary, limited-access AWS credentials directly to the client. This eliminates the need to build a credentials broker for permissions to access your AWS resources. For example, you might need to let users sign in with their social media accounts and access app assets from Amazon S3 for your mobile application. Identity pools also deliver credentials to users who sign in with user pools.

In this tutorial, you'll create a web application where you can obtain temporary authenticated and guest credentials in the enhanced and basic authentication flows with supported identity providers (IdPs) in identity pools. If you're already experienced in web development, download the example app from GitHub.

Download the example application from GitHub

This example application demonstrates the following capabilities of Amazon Cognito identity pools:

Authentication flows in identity pools

- Enhanced authentication flow with detailed API request breakdowns
- Basic authentication flow with detailed API request breakdowns

Implementing guest (unauthenticated) access

- Provide limited AWS service access without requiring sign-in

Integration with supported identity providers

- Social IdPs (Facebook, Amazon, Twitter, Apple, and Google) for consumer access
- Enterprise IdPs (through OpenID Connect or SAML) for corporate users
- Amazon Cognito user pools

AWS credentials management

- Exchanging identity provider tokens for temporary AWS credentials
- Using temporary credentials to access AWS services securely

After you set up the application on your development webserver and access it in a browser, you see the following options.

Topics

- Prerequisites
- Authentication provider setup
- Deploy the demo application
- Explore authentication methods in your identity pool
- Next steps

Prerequisites

Before you begin, you'll need the following resources configured.

- An AWS account with access to Amazon Cognito. If you do not have an AWS account, follow the instructions in Getting started with AWS.
- Python 3.11 or later installed on your development machine.
- GitHub access.
- AWS credentials configured with permissions to make authenticated requests to Amazon Cognito APIs. These credentials are required for developer authentication.

For more information about implementing AWS credentials and identity pool federation in your specific SDK, see the section called “Getting credentials”.

Authentication provider setup

For best results with this application, set up and integrate one or more third-party identity providers (IdPs) or Amazon Cognito user pools with your Amazon Cognito identity pool. After you complete the prerequisites and before you run this demo application, choose which identity providers to configure. The Amazon Cognito console walks you through the process of configuring identity pools and providers.

Amazon Cognito user pools

- the section called “Authentication”
- the section called “App clients”

Social identity providers

- Google: the section called “Google”
- Facebook: the section called “Facebook”
- Amazon: the section called “Login with Amazon”

OpenID Connect (OIDC) providers

- the section called “Open ID Connect providers”

SAML providers

- the section called “SAML identity providers”

> **Note**
>
> For this demo application, you don't need to set up all supported identity providers. You can start with one that matches your use case. Each link provides detailed configuration instructions.

Deploy the demo application

Clone the repository

1. Open a terminal window.
2. Clone the `aws-doc-sdk-examples` repository or otherwise retrieve this folder in the repo.

```
git clone https://github.com/awsdocs/aws-doc-sdk-examples.git
```

3. Navigate to the project directory.

```
cd python/example_code/cognito/scenarios/identity_pools_example_demo/web
```

Create an identity pool

To create an Amazon Cognito identity pool for your application, follow the instructions in the section called "Configuring identity pools".

To configure an identity pool for the demo application

1. Open the Amazon Cognito console.
2. From the left navigation menu, choose **Identity pools**. Choose an existing identity pool, or create a new one.
3. Under **User access**, enable **Authenticated access** and **Guest access**. Configure a new or existing IAM role and assign it the permissions that you want to grant to each type of user.
4. Under **User access**, set up any identity providers that you want to configure.
5. Under **Identity pool properties**, enable **Basic (classic) authentication**.
6. Keep your browser open to the console for your identity pool. You will use the identity pool ID and other configuration information in your application setup.

Configure and run the application

The following steps guide you through initial setup of your demo application.

To configure the demo application

1. Open a command line in `python/example_code/cognito/scenarios/identity_pools_example_demo/web` in your `aws-doc-sdk-examples` clone.
2. Create a `.env` file by copying the example environment file.

```
cp .env.example .env
```

3. Open the `.env` file in a text editor. Replace the example values in the file with your own configuration values.

4. Install backend dependencies.

```
pip install -r requirements.txt
```

5. Start the backend server:

```
cd backend
python oauth_server.py
```

6. Open a new terminal window, navigate to the project directory and start the frontend server:

```
cd frontend
python -m http.server 8001
```

7. Open your browser to the application at http://localhost:8001. Your browser will display the demo application interface, ready for testing identity pools authentication.

Explore authentication methods in your identity pool

This section guides you through basic and enhanced authentication flows using the Amazon Cognito identity pools demo application. With this demo, you'll learn how identity pools work with various identity providers to provide temporary AWS credentials to your application users.

In the **Interactive demo** section of the example application, you will first choose between two types of access supported by identity pools.

Unauthenticated (guest) access

Provide AWS credentials to users who haven't yet authenticated.

Authenticated access

Exchange identity provider tokens for AWS credentials with a full scope of available permissions. Choose an identity provider from among those that you configured in your `.env` file.

Unauthenticated (guest) access

This step demonstrates how to obtain temporary AWS credentials for unauthenticated (guest) users through your identity pool's guest access feature. In the demo application, you'll test both enhanced and basic flows to see how identity pools issue credentials without requiring user sign-

in. Guest access uses the same API sequence as authenticated access, but without providing identity provider tokens (such as OAuth tokens from Google, Facebook, or SAML assertions from enterprise providers).

Continue reading if you are looking for information about providing limited AWS access to users without requiring authentication. After you implement guest access, you'll learn how to securely provide AWS credentials to anonymous users and understand the differences between the two authentication flows.

> **Important**
>
> Unauthenticated access can issue credentials to anyone with internet access, so it's best used for AWS resources that require minimal security, such as public APIs and graphics assets. Before proceeding with this step, check if you've configured your identity pool with guest access enabled and ensure proper IAM policies are in place to limit permissions.

Guest access with enhanced flow

The enhanced flow is a streamlined approach to obtaining AWS credentials for unauthenticated users with two API requests.

To test guest access with the enhanced flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Guest access** tab.
3. Choose the **Enhanced flow** tab.
4. Choose **Test guest access**.
5. The app obtains temporary AWS credentials from your identity pools without additional authentication prompts.
6. After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
```

```
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-07T00:58:21-07:00"
  }
}
```

b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- `GetId()` API request with your `identityPoolId`. No authentication tokens required for guest access

```
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

If valid, it finds or creates and returns the user's `IdentityID`. An example response looks like this:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

- `GetCredentialsForIdentity()` with the returned `identityPoolId`.

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

Cognito validates guest access, assumes the unauthenticated role internally with AWS STS, and return temporary AWS credential. (No IAM authentication on this call; role trust must allow `cognito-identity-amazonzaws.com`.)

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
```

```
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-07T00:58:21-07:00"
  }
}
```

Guest access with basic flow

The basic flow provides granular control over the authentication process with separate API requests for identity retrieval and credential generation.

To test guest access with the basic flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Guest access** tab.
3. Choose the **Basic flow** tab.
4. Choose **Test guest access**.
5. The app obtains temporary AWS credentials from your identity pools without additional authentication prompts.
6. After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them.

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
     "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
     "Credentials": {
       "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
       "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
       "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
       "Expiration": "2025-08-12T13:36:17-07:00"
     }
   }
   ```

   b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

   - `GetId()` API request with your identity pool ID. No authentication tokens required for guest access.

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111"
}
```

If valid, it finds or creates and returns the user's `IdentityID`. An example response looks like this:

```
{
   "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetOpenIdToken()` with the returned `IdentityID` and the same `Logins` map

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE......"
}
```

What happens in this step: Amazon Cognito issues a short-lived OpenID Connect web identity token from cognito-identity.amazonaws.com that represents this `IdentityId`. The token includes OIDC claims that AWS STS evaluates, including aud (your identity pool ID) and amr (authenticated or unauthenticated). Your IAM role's trust policy must require those claims.

- `AssumeRoleWithWebIdentity()` - Your app calls AWS STS directly to exchange the Amazon Cognito OpenID token for temporary AWS credentials

```
POST sts:AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/
Cognito_IdentityPoolUnauth_Role",
```

```
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE......"
}
```

Response:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXdzEEXAMPLE......"
  }
}
```

What happens in this step: Once validated: returns temporary AWS credentials

Use the temporary credentials

These temporary credentials function as standard AWS credentials but with limited permissions defined by your identity pool's unauthenticated IAM role. You can use them with any AWS SDK or AWS CLI. For more information about configuring AWS SDKs with credentials, see Standardized credentials providers in the AWS SDKs and Tools Reference Guide.

The examples below are not a complete list, but they show common ways an identity pool's guest feature can improve user experience.

Public, read-only content

The following examples configure credentials providers for limited Amazon S3 access as a guest user.

Python

```
# Example: Using credentials with boto3
import boto3

# Configure client with temporary credentials
s3_client = boto3.client(
    's3',
    aws_access_key_id='AKIAIOSFODNN7EXAMPLE',
    aws_secret_access_key='wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY',
    aws_session_token='IQoJb3JpZ2luX2VjEEXAMPLE......'
```

```
)

# Make API requests within IAM role permissions
response = s3_client.list_objects_v2(Bucket='my-public-bucket')

# Access public content
for obj in response.get('Contents', []):
    print(f"File: {obj['Key']}, Size: {obj['Size']} bytes")
```

JavaScript

```
// Example: Accessing public content
import { S3Client, GetObjectCommand } from "@aws-sdk/client-s3";

const s3Client = new S3Client({
    region: "us-east-1",
    credentials: {
        accessKeyId: 'AKIAIOSFODNN7EXAMPLE',
        secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY',
        sessionToken: 'IQoJb3JpZ2luX2VjEEXAMPLE......'
    }
});

// Access public images or documents
const response = await s3Client.send(new GetObjectCommand({
    Bucket: 'my-public-content',
    Key: 'product-catalog.pdf'
}));
```

"Try-before-login" features

The following examples make use of read-only access to Amazon DynamoDB as a guest user.

Python

```
# Example: Limited app functionality for trial users
import boto3

dynamodb = boto3.client(
    'dynamodb',
    aws_access_key_id='AKIAIOSFODNN7EXAMPLE',
    aws_secret_access_key='wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY',
```

```
    aws_session_token='IQoJb3JpZ2luX2VjEEXAMPLE......'
)

# Allow guest users to view sample data (limited to 5 items)
response = dynamodb.scan(TableName='SampleProducts', Limit=5)
```

JavaScript

```
// Example: Limited app functionality for trial users
import { DynamoDBClient, ScanCommand } from "@aws-sdk/client-dynamodb";

const dynamodbClient = new DynamoDBClient({
    region: "us-east-1",
    credentials: {
        accessKeyId: 'AKIAIOSFODNN7EXAMPLE',
        secretAccessKey: 'wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY',
        sessionToken: 'IQoJb3JpZ2luX2VjEEXAMPLE......'
    }
});

// Allow guest users to view sample data (limited to 5 items)
const response = await dynamodbClient.send(new ScanCommand({
    TableName: 'SampleProducts',
    Limit: 5
}));
```

Social identity provider authentication

This step explores the overall flow for using social identity providers with Amazon Cognito identity pools. Social authentication provides a familiar sign-in experience while maintaining security through federated identity management. You can sign in from a social identity provider (IdP) like Google, Facebook, and Amazon, and then exchange that IdP token for temporary AWS credentials. Twitter and Apple integration is also supported by identity pools, but isn't supported in the example application.

The identity pool itself is not a user directory. It doesn’t store passwords or profile fields. Instead, it trusts external IdPs to authenticate the user and focuses on authorizing that already-authenticated user to directly call AWS services, by vending credentials for IAM roles.

Social identity provider with enhanced flow

This section shows how you can use a social identity provider to sign in a user and, using the enhanced flow, exchange the provider token in an Amazon Cognito identity pool for temporary credentials to request AWS resources.

Use social sign-in with the enhanced flow in the example application

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Enhanced flow** tab.
4. Choose a supported social provider you've configured, for example **Sign in with Google**, **Sign in with Facebook**, or **Login with Amazon**.
5. Sign in and consent to share user data with the application.
6. The provider redirects back to the app's redirect URI
7. The app sends the provider token to your identity pool and retrieves temporary AWS credentials
8. The app displays the **Results** panel in web interface.

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
     "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
     "Credentials": {
       "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
       "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
       "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
       "Expiration": "2025-08-12T13:36:17-07:00"
     }
   }
   ```

   b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- The app signs in the user with a social IdP and obtains the provider token. Identity pools accept these artifacts from social providers:

Identity provider	Cognito provider key	Purpose
Google	`accounts.google.com`	OAuth 2.0 tokens from Google Sign-In
Facebook	`graph.facebook.com`	Access tokens from Facebook Login
Amazon	`www.amazon.com`	OAuth tokens from Login with Amazon

After successful authentication with the social provider, your app receives an OAuth response containing the access token and other authentication details:

```
{
   "access_token": "ya29.A0AS3H6NEXAMPLE......",
   "expires_in": 3599,
   "scope": "openid https://www.examplesocial....",
   "token_type": "Bearer",
   "id_token": "eyJhbGciOiJSUzI1NiIsEXAMPLE......"
}
```

- `GetId()` API request with your identity pool ID and a `Logins`map containing your social provider token

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "accounts.google.com": "eyJhbGciOiJSUzI1NiIsEXAMPLE......"
  }
}
```

Response:

```
{
```

```
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetCredentialsForIdentity()` with the returned `IdentityID` and the same `Logins` map

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "accounts.google.com": "eyJhbGciOiJSUzI1NiIsEXAMPLE......"
  }
}
```

Response:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-07T00:58:21-07:00"
  },
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

What happened: Amazon Cognito validated the token against the configured provider, chose an IAM role based on your provider configuration, and called AWS STS on your behalf. Your identity pool then returned temporary credentials.

Social identity provider with basic flow

This section shows how you can use a social identity provider to sign in a user and, using the basic flow, exchange the provider token in an Amazon Cognito identity pool for temporary credentials to call AWS services.

Use social sign-in with the basic flow in the example application

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.

3. Choose the **Basic flow** tab.
4. Choose a supported social provider you've configured, for example **Sign in with Google**, **Sign in with Facebook**, or **Login with Amazon**.
5. Sign in and consent to share user data with the application.
6. The provider redirects back to the app's redirect URI
7. The app sends the provider token to your identity pool and retrieves temporary AWS credentials
8. The app displays the **Results** panel in web interface.

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
     "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
     "Credentials": {
       "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
       "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
       "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
       "Expiration": "2025-08-12T13:36:17-07:00"
     }
   }
   ```

   b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

   - The app signs in the user with a social IdP and obtains the provider token. Identity pools accept these artifacts from social providers:

Identity provider	Cognito provider key	Purpose
Google	`accounts.google.com`	OAuth 2.0 tokens from Google Sign-In
Facebook	`graph.facebook.com`	Access tokens from Facebook Login

Identity provider	Cognito provider key	Purpose
Amazon	`www.amazon.com`	OAuth tokens from Login with Amazon

After successful authentication with the social provider, your app receives an OAuth response containing the access token and other authentication details:

```
{
   "access_token": "ya29.A0AS3H6NEXAMPLE......",
   "expires_in": 3599,
   "scope": "openid https://www.examplesocial....",
   "token_type": "Bearer",
   "id_token": "eyJhbGciOiJSUzI1NiIsEXAMPLE......"
}
```

- `GetId()` API request with your identity pool ID and a `Logins` map containing your social provider token

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "accounts.google.com": "token..."
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetOpenIdToken()` with the returned `IdentityID` and the same Logins map

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "accounts.google.com": "token..."
```

```
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE......"
}
```

- `AssumeRoleWithWebIdentity()` with the OpenID token

```
POST AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE......"
}
```

Response:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-12T14:36:17-07:00"
  }
}
```

What happened: Amazon Cognito validated the token against the configured provider and issued an OpenID token. The application called AWS STS directly to assume an IAM role and receive temporary credentials.

Understand social access

- Social users receive temporary AWS credentials through Amazon Cognito identity pools after authenticating with their social provider.
- Each authenticated user gets a unique identity ID that persists across sessions.

- These credentials are linked to an IAM role specifically designed for authenticated access, providing broader permissions than guest access.
- Social provider tokens are exchanged for AWS credentials, maintaining user identity and permissions.

Amazon Cognito user pools authentication

This step explores Amazon Cognito authentication with user pool managed login integration. When you link a user pool as an IdP to an identity pool, user pool tokens authorize your identity pool to issue temporary credentials.

User pool authentication with enhanced flow

The enhanced flow provides a streamlined approach to obtaining AWS credentials through Amazon Cognito identity pools with a single API request.

Use Amazon Cognito user pool authentication with the identity pool enhanced flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Enhanced flow** tab.
4. Choose **Sign in with Amazon Cognito user pools**
5. Complete sign-in with your username and password in managed login.
6. The user pool redirects back to your application redirect URI with an authorization code.
7. The application exchanges the authorization code with your user pool for JSON web tokens.
8. The application exchanges the ID token with your identity pool for temporary AWS credentials
9. The app displays the **Results** panel in web interface

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

```
{
```

```
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- The app signs in the user with a Amazon Cognito. After successful authentication with the user pool, your app receives an OAuth 2.0 response containing the ID token (JWT). Identity pools accept JWT ID tokens from user pools using this provider key format:

Identity provider	Cognito provider key	Purpose
Amazon Cognito user pool	`cognito-idp.{region}.amazonaws.com/{user-pool-id}`	JWT ID tokens from Amazon Cognito user pools

After successful authentication with the user pool, your app receives an OAuth 2.0 response containing the ID token (JWT):

```
{
   "id_token": "eyJraWQiOiJFWAMPLE......",
   "token_type": "Bearer",
   "expires_in": 3600
}
```

- `GetId()` API request with your `identityPoolId`and a `Logins` map that includes your user-pool provider key mapped to the `id_token`. Amazon Cognito verified the user pool ID token's signature, issuer, expiry, and audience (aud) matches one of the app client IDs you registered for this user pool IdP in the identity pool.

```
POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:1ac4a76d-1fef-48aa-83af-4224799c0b5c",
```

```
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
 "eyJraWQiOiJFWAMPLE......"
  }
}
```

If valid, it finds or creates and returns the user's `IdentityID`. An example response looks like this:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetCredentialsForIdentity()` with the returned `identityPoolId`and a=the same `Logins` map with the `id_token`. Amazon Cognito revalidates the user pool ID token's signature, issuer, expiry, and audience (aud) matches one of the app client IDs you registered for this user pool IdP in the identity pool.

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
 "eyJraWQiOiJFWAMPLE......"
  }
}
```

If valid, it chooses an IAM role (roles-in-token, rules, or default), calls AWS STS on your behalf, and returns temporary AWS credentials. An example response looks like this:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "ASIAW7TIP7EJEXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-12T14:36:17-07:00"
  }
}
```

User pool authentication with basic flow

The basic flow provides granular control over the authentication process with separate API requests for identity retrieval and credential generation.

Use Amazon Cognito user pool authentication with the identity pool basic flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Basic flow** tab.
4. Choose **Sign in with Amazon Cognito user pools**
5. Complete sign-in with your username and password in managed login.
6. The user pool redirects back to your application redirect URI with an authorization code.
7. The application exchanges the authorization code with your user pool for JSON web tokens.
8. The application exchanges the ID token with your identity pool for temporary AWS credentials
9. The app displays the **Results** panel in web interface

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
     "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
     "Credentials": {
       "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
       "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
       "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
       "Expiration": "2025-08-12T13:36:17-07:00"
     }
   }
   ```

   b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- The app signs in the user with a Amazon Cognito user pool and obtains ID token (JWT) as the artifact. After successful authentication with the user pool, your app receives an OAuth response containing the ID token (JWT). Identity pools use this token for authentication:

```
{
   "id_token": "eyJraWQiOiJFWAMPLE......",
   "token_type": "Bearer",
   "expires_in": 3600
}
```

- `GetId()` API request with your identity pool ID and a `Logins` map that includes your user-pool provider key and the ID token as the value. Amazon Cognito verified the user pool ID token's signature, expiry, and audience (aud) matches one of the app client IDs you registered for this user pool IdP in the identity pool.

```
POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:1ac4a76d-1fef-48aa-83af-4224799c0b5c",
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
 "eyJraWQiOiJFWAMPLE......"
  }
}
```

  If valid, it finds or creates and returns the user's `IdentityID`. An example response looks like this:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetOpenIdToken()` with the returned `IdentityID` and the same `Logins` map

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "cognito-idp.us-east-1.amazonaws.com/us-east-1_EXAMPLE123":
 "eyJraWQiOiJFWAMPLE......"
```

```
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE......"
}
```

What happens in this step: Amazon Cognito issues a short-lived OpenID Connect web identity token from cognito-identity.amazonaws.com that represents this `IdentityId`. The token includes OIDC claims that AWS STS evaluates, including aud (your identity pool ID) and amr (authenticated or unauthenticated). Your IAM role's trust policy must require those claims.

- `AssumeRoleWithWebIdentity()` - Your app calls AWS STS directly to exchange the Amazon Cognito OpenID token for temporary AWS credentials

```
POST sts:AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE......",
  "RoleSessionName": "CognitoIdentityCredentials"
}
```

Response:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXdzEEXAMPLE......",
    "Expiration": "2025-08-12T14:36:17-07:00"
  },
  "AssumedRoleUser": {
    "AssumedRoleId": "AROAW7TIP7EJYEXAMPLE:CognitoIdentityCredentials",
    "Arn": "arn:aws:sts::111122223333:assumed-role/
Cognito_IdentityPoolAuth_Role/CognitoIdentityCredentials"
  }
}
```

What your demo application did: Your app sent the OpenID token from `GetOpenIdToken()` to AWS STS, requesting temporary credentials. AWS STS performed validation checks and issued credentials:

Understand user pool access

- User pool users receive temporary AWS credentials through Amazon Cognito identity pools.
- These credentials are linked to an IAM role specified in your identity pool configuration.
- User pool ID tokens are exchanged for AWS credentials through the identity pool.

SAML authentication

This step explores SAML authentication. Users can sign in with enterprise identity providers that support SAML to access AWS services. The basic flow with SAML isn't supported in the example application.

SAML authentication with enhanced flow

This section shows how you can use a SAML identity provider to sign in a user and, using the enhanced flow, exchange the SAML assertion in an Amazon Cognito identity pool for temporary AWS credentials to call AWS services.

Use SAML authentication with the identity pool enhanced flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Enhanced flow** tab.
4. Choose **Sign in with SAML provider**
5. Complete sign-in with your enterprise credentials.
6. The user pool redirects back to your application redirect URI with a SAML assertion.
7. The application exchanges the authorization code with your user pool for JSON web tokens.
8. The application exchanges the SAML response with your identity pool for temporary AWS credentials
9. The app displays the **Results** panel in web interface

After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- The app signs in the user with a SAML IdP and obtains the SAML response. Identity pools accept SAML assertions from enterprise providers using the SAML provider ARN as the key:

Identity provider	Cognito provider key	Purpose
SAML Provider	`arn:aws:iam::111122223333:saml-provider/EXAMPLE`	SAML assertions from enterprise IdPs

After successful authentication with the SAML provider, your app receives a SAML response via HTTP POST to your callback URL:

```
{
  "saml_response": "PD94bWwgdmVyc2lvbj0iMS4wIiBFWAMPLE...",
  "provider_arn": "arn:aws:iam::111122223333:saml-provider/EXAMPLE",
  "status": "Authentication successful"
}
```

- `GetId()` API request with your identity pool ID and a `Logins` map containing your SAML provider ARN and assertion

```
POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "arn:aws:iam::111122223333:saml-provider/EXAMPLE":
 "PD94bWwgdmVyc2lvbj0iMS4wIiBFWAMPLE..."
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetCredentialsForIdentity()` with the returned `IdentityID`and the same `Logins` map

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "arn:aws:iam::111122223333:saml-provider/EXAMPLE":
 "PD94bWwgdmVyc2lvbj0iMS4wIiBFWAMPLE..."
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......"
  }
```

```
}
```

What happened: Amazon Cognito validated the SAML assertion against the configured provider, chose an IAM role based on SAML attributes or rules, and called AWS STS on your behalf.

Understand SAML access

- Enterprise users receive temporary AWS credentials from Amazon Cognito identity pools after authenticating with their SAML provider.
- Each authenticated user gets a unique identity ID that persists across sessions.
- These credentials are linked to an IAM role specifically designed for authenticated access, providing broader permissions than guest access.
- SAML assertions are exchanged for AWS credentials, maintaining user identity and enterprise attributes.

OpenID Connect (OIDC) authentication

This step explores OIDC authentication with enterprise identity providers. Users can sign in through their organization's enterprise identity provider (such as Azure AD, Okta, or Google Workspace) to access AWS services. Continue reading if you are looking for information about integrating standards-based authentication with your AWS resources. After you implement OIDC authentication, you'll learn how to leverage OIDC claims for fine-grained access control.

OIDC authentication with enhanced flow

This section shows how you can use an OIDC identity provider to sign in a user and, using the enhanced flow, exchange the OIDC token in an Amazon Cognito identity pool for temporary AWS credentials to call AWS services.

Use OIDC sign-in with the identity pool enhanced flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Enhanced flow** tab.
4. Choose **Sign in with OIDC provider**

5. Complete sign-in with your enterprise credentials.
6. The OIDC provider redirects back to the app with an authorization code
7. The application exchanges the authorization code with your user pool for JSON web tokens.
8. The application sends the OIDC token to your identity pool and retrieves temporary AWS credentials.
9. The app displays the **Results** panel in web interface

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
     "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
     "Credentials": {
       "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
       "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
       "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
       "Expiration": "2025-08-12T13:36:17-07:00"
     }
   }
   ```

   b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

   - The app signs in the user with an OIDC IdP and obtains the ID token. Identity pools accept OIDC tokens from enterprise providers:

Identity provider	Cognito provider key	Purpose
OIDC Provider	`example-provider.com/oauth2/default`	OIDC ID tokens from enterprise IdPs

   After successful authentication with the OIDC provider, your app receives an OAuth 2.0 response containing the tokens:

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "eyJraWQiOiJFWAMPLE......",
  "scope": "email openid profile",
  "id_token": "eyJraWQiOiJFWAMPLE......"
}
```

- `GetId()` API request with your identity pool ID and a `Logins` map containing your OIDC provider token

```
POST GetId
{
  "AccountId": "111122223333",
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE......"
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetCredentialsForIdentity()` with the returned `IdentityID` and the same Logins map

```
POST GetCredentialsForIdentity
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE......"
  }
}
```

Response:

```
{
```

```
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......"
  }
}
```

What happened: Amazon Cognito validated the OIDC token against the configured provider, chose an IAM role (default, claim-based, or rule-mapped) and called AWS STS on your behalf.

OIDC authentication with basic flow

This section shows how you can use an OIDC identity provider to sign in a user and, using the basic flow, exchange the OIDC token in an Amazon Cognito identity pool for temporary AWS credentials to call AWS services.

Use OIDC sign-in with the identity pool basic flow

1. In the demo application, navigate to **Interactive demo** section
2. Choose the **Authenticated access** tab.
3. Choose the **Basic flow** tab.
4. Choose **Sign in with OIDC provider**
5. Complete sign-in with your enterprise credentials.
6. The OIDC provider redirects back to the app with an authorization code
7. The application exchanges the authorization code with your user pool for JSON web tokens.
8. The application sends the OIDC token to your identity pool and retrieves temporary AWS credentials.
9. The app displays the **Results** panel in web interface

   After successful authentication, you will see the web interface displaying the **Results** panel, and you have two options to explore them:

   a. **View credentials only** button: Choose this button if you want to directly see temporary AWS credentials generated without the API flow details.

   ```
   {
   ```

```
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "IQoJb3JpZ2luX2VjEEXAMPLE......",
    "Expiration": "2025-08-12T13:36:17-07:00"
  }
}
```

b. **View detailed API flow** button: Choose this button if you want to see the step-by-step API requests.

- The app signs in the user with an OIDC IdP and obtains the ID token. Identity pools accept OIDC tokens from enterprise providers:

Identity provider	Cognito provider key	Purpose
OIDC Provider	`example-provider.com/oauth2/default`	OIDC ID tokens from enterprise IdPs

After successful authentication with the OIDC provider, your app receives an OAuth 2.0 response containing the tokens:

```
{
  "token_type": "Bearer",
  "expires_in": 3600,
  "access_token": "eyJraWQiOiJFWAMPLE......",
  "scope": "openid email profile",
  "id_token": "eyJraWQiOiJFWAMPLE......"
}
```

- `GetId()` API request with your identity pool ID and a `Logins`map containing your OIDC provider token

```
POST GetId
{
  "IdentityPoolId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE......"
  }
```

```
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222"
}
```

- `GetOpenIdToken()` with the returned IdentityID and the same `Logins`map

```
POST GetOpenIdToken
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Logins": {
    "example-provider.com/oauth2/default": "eyJraWQiOiJFWAMPLE......"
  }
}
```

Response:

```
{
  "IdentityId": "us-east-1:a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
  "Token": "eyJraWQiOiJFWAMPLE......"
}
```

- `AssumeRoleWithWebIdentity()` with the OpenID token

```
POST AssumeRoleWithWebIdentity
{
  "RoleArn": "arn:aws:iam::111122223333:role/Cognito_IdentityPoolAuth_Role",
  "WebIdentityToken": "eyJraWQiOiJFWAMPLE......"
}
```

Response:

```
{
  "Credentials": {
    "AccessKeyId": "AKIAIOSFODNN7EXAMPLE",
    "SecretAccessKey": "wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY",
    "SessionToken": "FwoGZXIvYXdzEEXAMPLE......",
    "Expiration": "2025-08-12T14:36:17-07:00"
```

```
  }
}
```

What happened: Amazon Cognito validated the OIDC token against the configured provider and returned an OpenID token. The application called AWS STS directly to assume the appropriate IAM role and received short-lived credentials.

Understand OIDC authentication

- Standards-based: OIDC is built on OAuth 2.0 and provides standardized identity information.
- Token validation: ID tokens can be validated for authenticity.
- Claims-based access: OIDC claims can be used for role mapping and access control.
- Enterprise integration: Works with popular enterprise identity providers.

Next steps

Now that you've set up and explored the demo application, you can:

- Configure additional identity providers you haven't tested yet
- Experiment with both enhanced and basic authentication to understand their differences
- Customize the demo for your own use case
- Integrate Amazon Cognito Identity Pools into your own applications.

Security in Amazon Cognito

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from a data center and network architecture that is built to meet the requirements of the most security-sensitive organizations.

Security is a shared responsibility between AWS and you. The describes this as security of the cloud and security in the cloud:

- **Security of the cloud** – AWS is responsible for protecting the infrastructure that runs AWS services in the AWS Cloud. AWS also provides you with services that you can use securely. Third-party auditors regularly test and verify the effectiveness of our security as part of the .
- **Security in the cloud** – Your responsibility is determined by the AWS service that you use. You are also responsible for other factors including the sensitivity of your data, your company's requirements, and applicable laws and regulations

This documentation helps you understand how to apply the shared responsibility model when using Amazon Cognito. It shows you how to configure Amazon Cognito to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your Amazon Cognito resources.

For user pool–specific protections against malicious sign-up, sign-in, and account takeover—MFA, adaptive authentication, and threat protection—see Protecting user pools with threat protection.

Contents

- Data protection in Amazon Cognito
- Identity and access management for Amazon Cognito
- Logging and monitoring in Amazon Cognito
- Access Amazon Cognito using an interface endpoint (AWS PrivateLink)
- Compliance validation for Amazon Cognito
- Resilience in Amazon Cognito
- Infrastructure security in Amazon Cognito
- Configuration and vulnerability analysis in Amazon Cognito user pools
- AWS managed policies for Amazon Cognito

Data protection in Amazon Cognito

The AWS applies to data protection in Amazon Cognito (Amazon Cognito). As described in this model, AWS is responsible for protecting the global infrastructure that runs all of the AWS Cloud. You are responsible for maintaining control over your content that is hosted on this infrastructure. This content includes the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see the Data Privacy FAQ.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual user accounts with AWS Identity and Access Management (IAM). That way each user is given only the permissions necessary to fulfill their job duties. We also recommend that you secure your data in the following ways:

- Use multi-factor authentication (MFA) with each account.
- Use SSL/TLS to communicate with AWS resources.
- Set up API and user activity logging with AWS CloudTrail.
- Use AWS encryption solutions, along with all default security controls within AWS services.
- Use advanced managed security services such as Amazon Macie, which assists in discovering and securing personal data that is stored in Amazon S3.

We strongly recommend that you never put sensitive identifying information, such as your customers' account numbers, into free-form fields such as a **Name** field. This includes when you work with Amazon Cognito or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into Amazon Cognito or other services might get picked up for inclusion in diagnostic logs. When you provide a URL to an external server, don't include credentials information in the URL to validate your request to that server.

Data encryption

Data encryption typically falls into two categories: encryption at rest and encryption in transit.

Encryption at rest

Data within Amazon Cognito user pools and identity pools is encrypted at rest in accordance with industry standards.

- Amazon Cognito encrypts customer data in identity pools with AWS owned keys. You can't change this behavior.

- *By default*, Amazon Cognito also encrypts customer data in user pools with AWS owned keys. You can also configure your user pools to instead encrypt your customers' information with customer managed keys.

AWS owned key

Amazon Cognito encrypts the data in your user pool or identity pool with an AWS owned KMS key. Keys of this type aren't visible in AWS KMS.

Customer managed key

Amazon Cognito encrypts the data in your user pool with a customer managed key. You own the administration of customer managed key policies, rotation, and scheduled deletion.

> **Note**
>
> Encryption with customer managed keys might not be available in some user pools. Newly-created user pools always have this form of encryption available to them.

Things to know about user pool encryption with customer managed keys

1. All customer data in user pools is encrypted at rest, even if you take no action to configure encryption settings.
2. Amazon Cognito supports only symmetric KMS keys in the same AWS Region as your user pool for user pool encryption at rest. You can't configure user pool encryption at rest with asymmetric keys. You can configure encryption at rest with single-Region keys and with multi-Region keys that are in the same Region as your user pool.
3. You can configure user pool encryption only with a KMS key ARN, not an alias.

PII encryption

Amazon Cognito supports the confidentiality, integrity, and availability of personally identifiable information (PII) in user attribute searches with searchable encryption. These Hash-based Message Authentication Code (HMAC) functions, performance-optimized for user pool datasets, map between the plaintext and encrypted values of user attributes. Amazon Cognito calculates HMAC values with the KMS key that encrypts your user pool. This protection applies to the following attributes:

- `sub`
- `email`
- `phone_number`
- `given_name`
- `family_name`
- `name`
- `username`
- `preferred_username`
- `cognito:user_status`

The following procedures configure encryption at rest in your user pool. For more information about KMS key policies that delegate access to AWS services like Amazon Cognito, see Permissions for Amazon Cognito in key policies.

Set customer managed key policy

To use a customer managed key, your key must trust an Amazon Cognito service principal to perform encryption and decryption operations on the key. Configure the key policy of your KMS key as shown in the following example. The IAM principal that writes this policy must have write access to your KMS key, with `kms:PutKeyPolicy` permission.

```
{
    "Id": "cognito-cmk-policy",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Amazon Cognito service access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.amazonaws.com",
                    "identitystore.amazonaws.com"
                ]
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
```

```
                "kms:GenerateDataKeyWithoutPlainText"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                    ]
                },
                "StringEquals": {
                    "aws:SourceAccount": [
                        "111122223333"
                    ]
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:cognito-idp:userpool-arn":
 "arn:aws:cognito-idp:eusc-de-east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                }
            }
        },
        {
            "Sid": "Allow Amazon Cognito service DescribeKey access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.amazonaws.com",
                    "identitystore.amazonaws.com"
                ]
            },
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                    ]
                },
                "StringEquals": {
                    "aws:SourceAccount": [
                        "111122223333"
```

```
                    ]
                }
            }
        },
        {
            "Sid": "Allow access through Amazon Cognito for all principals in
account that are authorized to use Amazon Cognito",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "cognito-idp.eusc-de-east-1.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "Allow access through Amazon Cognito for all principals in
account that are authorized to use Amazon Cognito",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKeyWithoutPlainText"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "cognito-idp.eusc-de-east-1.amazonaws.com"
                    ]
                },
                "StringLike": {
```

```
                    "kms:EncryptionContext:aws:cognito-idp:userpool-arn":
 "arn:aws:cognito-idp:eusc-de-east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                }
            }
        },
        {
            "Sid": "Allow administrators to manage the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

Set customer managed key policy with permissions for encryption of exported logs

Apply this policy to your KMS key when you have log export configured. This policy permits Amazon Cognito to encrypt logs as it exports them, and the intermediate service to decrypt them before writing them to your CloudWatch Logs log group. Configure the key policy of your KMS key as shown in the following example. The IAM principal that writes this policy must have write access to your KMS key, with `kms:PutKeyPolicy` permission.

```
{
    "Id": "cognito-cmk-policy",
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "Allow Amazon Cognito service access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.amazonaws.com",
                    "identitystore.amazonaws.com"
                ]
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKeyWithoutPlainText"
```

```
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                    ]
                },
                "StringEquals": {
                    "aws:SourceAccount": [
                        "111122223333"
                    ]
                },
                "StringLike": {
                    "kms:EncryptionContext:aws:cognito-idp:userpool-arn":
 "arn:aws:cognito-idp:eusc-de-east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                }
            }
        },
        {
            "Sid": "Allow Amazon Cognito service DescribeKey access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.amazonaws.com",
                    "identitystore.amazonaws.com"
                ]
            },
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "ArnEquals": {
                    "aws:SourceArn": [
                        "arn:aws:cognito-idp:eusc-de-
east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                    ]
                },
                "StringEquals": {
                    "aws:SourceAccount": [
                        "111122223333"
                    ]
```

```
                }
            }
        },
        {
            "Sid": "Allow access through Amazon Cognito for all principals in
account that are authorized to use Amazon Cognito",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:DescribeKey"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "cognito-idp.eusc-de-east-1.amazonaws.com"
                    ]
                }
            }
        },
        {
            "Sid": "Allow access through Amazon Cognito for all principals in
account that are authorized to use Amazon Cognito",
            "Effect": "Allow",
            "Principal": {
                "AWS": "*"
            },
            "Action": [
                "kms:Encrypt",
                "kms:Decrypt",
                "kms:ReEncrypt*",
                "kms:GenerateDataKeyWithoutPlainText"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "kms:ViaService": [
                        "cognito-idp.eusc-de-east-1.amazonaws.com"
                    ]
                },
                "StringLike": {
```

```
                    "kms:EncryptionContext:aws:cognito-idp:userpool-arn":
 "arn:aws:cognito-idp:eusc-de-east-1:111122223333:userpool/eusc-de-east-1_EXAMPLE"
                }
            }
        },
        {
            "Sid": "Allow Amazon Cognito service log delivery access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "cognito-idp.amazonaws.com"
                ]
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Encrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:SourceArn": "arn:aws:logs:eusc-de-
east-1:111122223333:*"
                }
            }
        },
        {
            "Sid": "Allow IngestionHub service log delivery access",
            "Effect": "Allow",
            "Principal": {
                "Service": [
                    "delivery.logs.amazonaws.com"
                ]
            },
            "Action": [
                "kms:GenerateDataKey",
                "kms:Decrypt"
            ],
            "Resource": "*",
            "Condition": {
                "StringLike": {
                    "kms:EncryptionContext:SourceArn": "arn:aws:logs:eusc-de-
east-1:111122223333:*"
                }
            }
```

```
        },
        {
            "Sid": "Allow administrators to manage the key",
            "Effect": "Allow",
            "Principal": {
                "AWS": "arn:aws:iam::111122223333:root"
            },
            "Action": "kms:*",
            "Resource": "*"
        }
    ]
}
```

Configure encryption at rest in the console

To configure encryption at rest in a user pool

1. Go to the Amazon Cognito console. You might be prompted for your AWS credentials.
2. Choose **User Pools**.
3. Choose an existing user pool from the list, or create a user pool.
4. Choose the **Settings** menu and navigate to the **User pool security** tab. Locate **Encryption at rest** and select **Edit**.
5. Under **Key type**, select either **AWS owned key** or **Customer managed key**.
   a. If you selected **AWS owned key**, no additional configuration is required.
   b. If you selected **Customer managed key**, enter the ARN of a KMS key into **Customer managed key ARN**. You can also choose **Create an AWS KMS key** and open a new window in the AWS KMS console to create a new KMS key.
6. Choose **Save changes**.

Configure encryption at rest with the API

Set your key configuration in a CreateUserPool or UpdateUserPool API request. The following partial example request body sets a user pool to use the provided customer managed key. For a complete example request, see Examples.

```
"KeyConfiguration": {
   "KeyType": "CUSTOMER_MANAGED_KEY",
```

```
   "KmsKeyArn": "arn:aws:kms:us-east-1:111122223333:key/a1b2c3d4-5678-90ab-cdef-
EXAMPLE22222"
},
```

The following partial example request body sets a user pool to use an AWS owned key.

```
"KeyConfiguration": {
   "KeyType": "AWS_OWNED_KEY"
},
```

If your DescribeUserPool response doesn't include a `KeyConfiguration` parameter, your user pool is configured to encrypt data at rest with an AWS owned key.

Encryption in transit

As a managed service, Amazon Cognito is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Cognito through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Amazon Cognito user pools and identity pools have IAM-authenticated, unauthenticated, and token-authorized API operations. Some are *control plane*-type operations for administrative operations like configuring a user pool domain, and others are *data plane*-type operations for authentication. For more information, see List of API operations grouped by authorization model. All classes of Amazon Cognito API operations share a namespace—`cognito-idp` for user pools, `cognito-identity` for identity pools—and service endpoints. AWS service endpoints require encryption in transit with a minimum TLS version of 1.2.

Amazon Cognito user pools host managed login and the classic hosted UI on web domains that are served from service-owned Amazon CloudFront distributions. Amazon Cognito manages the

settings for encryption in transit on those distributions. For more information about the encryption settings for managed login, see TLS version in the managed login chapter.

Identity and access management for Amazon Cognito

AWS Identity and Access Management (IAM) is an AWS service that helps an administrator securely control access to AWS resources. IAM administrators control who can be *authenticated* (signed in) and *authorized* (have permissions) to use Amazon Cognito resources. IAM is an AWS service that you can use with no additional charge.

Topics

- Audience
- Authenticating with identities
- Managing access using policies
- How Amazon Cognito works with IAM
- Identity-based policy examples for Amazon Cognito
- Troubleshooting Amazon Cognito identity and access
- Using service-linked roles for Amazon Cognito

Audience

How you use AWS Identity and Access Management (IAM) differs based on your role:

- **Service user** - request permissions from your administrator if you cannot access features (see Troubleshooting Amazon Cognito identity and access)
- **Service administrator** - determine user access and submit permission requests (see How Amazon Cognito works with IAM)
- **IAM administrator** - write policies to manage access (see Identity-based policy examples for Amazon Cognito)

Authenticating with identities

Authentication is how you sign in to AWS using your identity credentials. You must be authenticated as the AWS account root user, an IAM user, or by assuming an IAM role.

You can sign in as a federated identity using credentials from an identity source like AWS IAM Identity Center (IAM Identity Center), single sign-on authentication, or Google/Facebook credentials. For more information about signing in, see How to sign in to your AWS account in the *AWS Sign-In User Guide*.

For programmatic access, AWS provides an SDK and CLI to cryptographically sign requests. For more information, see AWS Signature Version 4 for API requests in the *IAM User Guide*.

AWS account root user

When you create an AWS account, you begin with one sign-in identity called the AWS account *root user* that has complete access to all AWS services and resources. We strongly recommend that you don't use the root user for everyday tasks. For tasks that require root user credentials, see Tasks that require root user credentials in the *IAM User Guide*.

Federated identity

As a best practice, require human users to use federation with an identity provider to access AWS services using temporary credentials.

A *federated identity* is a user from your enterprise directory, web identity provider, or Directory Service that accesses AWS services using credentials from an identity source. Federated identities assume roles that provide temporary credentials.

For centralized access management, we recommend AWS IAM Identity Center. For more information, see What is IAM Identity Center? in the *AWS IAM Identity Center User Guide*.

IAM users and groups

An *IAM user* is an identity with specific permissions for a single person or application. We recommend using temporary credentials instead of IAM users with long-term credentials. For more information, see Require human users to use federation with an identity provider to access AWS using temporary credentials in the *IAM User Guide*.

An *IAM group* specifies a collection of IAM users and makes permissions easier to manage for large sets of users. For more information, see Use cases for IAM users in the *IAM User Guide*.

IAM roles

An *IAM role* is an identity with specific permissions that provides temporary credentials. You can assume a role by switching from a user to an IAM role (console) or by calling an AWS CLI or AWS API operation. For more information, see Methods to assume a role in the *IAM User Guide*.

IAM roles are useful for federated user access, temporary IAM user permissions, cross-account access, cross-service access, and applications running on Amazon EC2. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Managing access using policies

You control access in AWS by creating policies and attaching them to AWS identities or resources. A policy defines permissions when associated with an identity or resource. AWS evaluates these policies when a principal makes a request. Most policies are stored in AWS as JSON documents. For more information about JSON policy documents, see Overview of JSON policies in the *IAM User Guide*.

Using policies, administrators specify who has access to what by defining which **principal** can perform **actions** on what **resources**, and under what **conditions**.

By default, users and roles have no permissions. An IAM administrator creates IAM policies and adds them to roles, which users can then assume. IAM policies define permissions regardless of the method used to perform the operation.

Identity-based policies

Identity-based policies are JSON permissions policy documents that you attach to an identity (user, group, or role). These policies control what actions identities can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

Identity-based policies can be *inline policies* (embedded directly into a single identity) or *managed policies* (standalone policies attached to multiple identities). To learn how to choose between managed and inline policies, see Choose between managed policies and inline policies in the *IAM User Guide*.

Resource-based policies

Resource-based policies are JSON policy documents that you attach to a resource. Examples include IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-

based policies, service administrators can use them to control access to a specific resource. You must specify a principal in a resource-based policy.

Resource-based policies are inline policies that are located in that service. You can't use AWS managed policies from IAM in a resource-based policy.

Other policy types

AWS supports additional policy types that can set the maximum permissions granted by more common policy types:

- **Permissions boundaries** – Set the maximum permissions that an identity-based policy can grant to an IAM entity. For more information, see Permissions boundaries for IAM entities in the *IAM User Guide*.
- **Service control policies (SCPs)** – Specify the maximum permissions for an organization or organizational unit in AWS Organizations. For more information, see Service control policies in the *AWS Organizations User Guide*.
- **Resource control policies (RCPs)** – Set the maximum available permissions for resources in your accounts. For more information, see Resource control policies (RCPs) in the *AWS Organizations User Guide*.
- **Session policies** – Advanced policies passed as a parameter when creating a temporary session for a role or federated user. For more information, see Session policies in the *IAM User Guide*.

Multiple policy types

When multiple types of policies apply to a request, the resulting permissions are more complicated to understand. To learn how AWS determines whether to allow a request when multiple policy types are involved, see Policy evaluation logic in the *IAM User Guide*.

How Amazon Cognito works with IAM

Before you use IAM to manage access to Amazon Cognito, learn what IAM features are available to use with Amazon Cognito.

IAM features you can use with Amazon Cognito

IAM feature	Amazon Cognito support
Identity-based policies	Yes
Resource-based policies	No
Policy actions	Yes
Policy resources	Yes
Policy condition keys	Yes
ACLs	No
ABAC (tags in policies)	Partial
Temporary credentials	Yes
Principal permissions	No
Service roles	Yes
Service-linked roles	Yes

To get a high-level view of how Amazon Cognito and other AWS services work with most IAM features, see AWS services that work with IAM in the *IAM User Guide*.

Identity-based policies for Amazon Cognito

Supports identity-based policies: Yes

Identity-based policies are JSON permissions policy documents that you can attach to an identity, such as an IAM user, group of users, or role. These policies control what actions users and roles can perform, on which resources, and under what conditions. To learn how to create an identity-based policy, see Define custom IAM permissions with customer managed policies in the *IAM User Guide*.

With IAM identity-based policies, you can specify allowed or denied actions and resources as well as the conditions under which actions are allowed or denied. To learn about all of the elements that you can use in a JSON policy, see IAM JSON policy elements reference in the *IAM User Guide*.

Identity-based policy examples for Amazon Cognito

To view examples of Amazon Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito.

Resource-based policies within Amazon Cognito

Supports resource-based policies: No

Resource-based policies are JSON policy documents that you attach to a resource. Examples of resource-based policies are IAM *role trust policies* and Amazon S3 *bucket policies*. In services that support resource-based policies, service administrators can use them to control access to a specific resource. For the resource where the policy is attached, the policy defines what actions a specified principal can perform on that resource and under what conditions. You must specify a principal in a resource-based policy. Principals can include accounts, users, roles, federated users, or AWS services.

To enable cross-account access, you can specify an entire account or IAM entities in another account as the principal in a resource-based policy. For more information, see Cross account resource access in IAM in the *IAM User Guide*.

Policy actions for Amazon Cognito

Supports policy actions: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Action` element of a JSON policy describes the actions that you can use to allow or deny access in a policy. Include actions in a policy to grant permissions to perform the associated operation.

To see a list of Amazon Cognito actions, see Actions defined by Amazon Cognito in the *Service Authorization Reference*.

Policy actions in Amazon Cognito use the following prefix before the action:

```
cognito-identity
```

To specify multiple actions in a single statement, separate them with commas.

```
"Action": [
      "cognito-identity:action1",
      "cognito-identity:action2"
         ]
```

Signed versus unsigned APIs

When you sign Amazon Cognito API requests with AWS credentials, you can restrict them in an AWS Identity and Access Management (IAM) policy. API requests that you must sign with AWS credentials include server-side sign-in with `AdminInitiateAuth`, and actions that create, view, or modify your Amazon Cognito resources like `UpdateUserPool`. For more information about signed API requests, see Signing AWS API requests.

Because Amazon Cognito is a consumer identity product for apps that you want to make available to the public, you have access to the following unsigned APIs. Your app makes these API requests for your users and your prospective users. Some APIs require no prior authorization, like `InitiateAuth` to start a new authentication session. Some APIs use access tokens or session keys for authorization, like `VerifySoftwareToken` to complete MFA setup for a user that has an existing authenticated session. An unsigned, authorized Amazon Cognito user pools API supports a `Session` or `AccessToken` parameter in the request syntax as displayed in the Amazon Cognito API Reference. An unsigned Amazon Cognito Identity API supports an `IdentityId` parameter as displayed in the Amazon Cognito Federated Identities API Reference.

For more information about the authorization models and roles of Amazon Cognito user pools API operations, see List of API operations grouped by authorization model.

Amazon Cognito identity pools API operations

- `GetId`
- `GetOpenIdToken`
- `GetCredentialsForIdentity`
- `UnlinkIdentity`

Amazon Cognito user pools API operations

- `AssociateSoftwareToken`
- `ChangePassword`

- ConfirmDevice
- ConfirmForgotPassword
- ConfirmSignUp
- DeleteUser
- DeleteUserAttributes
- ForgetDevice
- ForgotPassword
- GetDevice
- GetUser
- GetUserAttributeVerificationCode
- GlobalSignOut
- InitiateAuth
- ListDevices
- ResendConfirmationCode
- RespondToAuthChallenge
- RevokeToken
- SetUserMFAPreference
- SetUserSettings
- SignUp
- UpdateAuthEventFeedback
- UpdateDeviceStatus
- UpdateUserAttributes
- VerifySoftwareToken
- VerifyUserAttribute

To view examples of Amazon Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito.

Policy resources for Amazon Cognito

Supports policy resources: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Resource` JSON policy element specifies the object or objects to which the action applies. As a best practice, specify a resource using its Amazon Resource Name (ARN). For actions that don't support resource-level permissions, use a wildcard (*) to indicate that the statement applies to all resources.

```
"Resource": "*"
```

Amazon resource names (ARNs)

ARNs for Amazon Cognito federated identities

In Amazon Cognito identity pools (federated identities), it is possible to restrict an IAM user's access to a specific identity pool, using the Amazon Resource Name (ARN) format, as in the following example. For more information about ARNs, see IAM identifiers.

```
arn:aws:cognito-identity:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

ARNs for Amazon Cognito Sync

In Amazon Cognito Sync, customers can also restrict access by the identity pool ID, identity ID, and dataset name.

For APIs that operate on an identity pool, the identity pool ARN format is the same as for Amazon Cognito Federated Identities, except that the service name is `cognito-sync` instead of `cognito-identity`:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID
```

For APIs that operate on a single identity, such as `RegisterDevice`, you can refer to the individual identity by the following ARN format:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/
identity/IDENTITY_ID
```

For APIs that operate on datasets, such as `UpdateRecords` and `ListRecords`, you can refer to the individual dataset using the following ARN format:

```
arn:aws:cognito-sync:REGION:ACCOUNT_ID:identitypool/IDENTITY_POOL_ID/
identity/IDENTITY_ID/dataset/DATASET_NAME
```

ARNs for Amazon Cognito user pools

For Amazon Cognito Your User Pools, it is possible to restrict a user's access to a specific user pool, using the following ARN format:

```
arn:aws:cognito-idp:REGION:ACCOUNT_ID:userpool/USER_POOL_ID
```

To see a list of Amazon Cognito resource types and their ARNs, see Resources defined by Amazon Cognito in the *Service Authorization Reference*. To learn with which actions you can specify the ARN of each resource, see Actions defined by Amazon Cognito.

To view examples of Amazon Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito.

Policy condition keys for Amazon Cognito

Supports service-specific policy condition keys: Yes

Administrators can use AWS JSON policies to specify who has access to what. That is, which **principal** can perform **actions** on what **resources**, and under what **conditions**.

The `Condition` element specifies when statements execute based on defined criteria. You can create conditional expressions that use condition operators, such as equals or less than, to match the condition in the policy with values in the request. To see all AWS global condition keys, see AWS global condition context keys in the *IAM User Guide*.

To see a list of Amazon Cognito condition keys, see Condition keys for Amazon Cognito in the *Service Authorization Reference*. To learn with which actions and resources you can use a condition key, see Actions defined by Amazon Cognito.

To view examples of Amazon Cognito identity-based policies, see Identity-based policy examples for Amazon Cognito.

Access control lists (ACLs) in Amazon Cognito

Supports ACLs: No

Access control lists (ACLs) control which principals (account members, users, or roles) have permissions to access a resource. ACLs are similar to resource-based policies, although they do not use the JSON policy document format.

Attribute-based access control (ABAC) with Amazon Cognito

Supports ABAC (tags in policies): Partial

Attribute-based access control (ABAC) is an authorization strategy that defines permissions based on attributes called tags. You can attach tags to IAM entities and AWS resources, then design ABAC policies to allow operations when the principal's tag matches the tag on the resource.

To control access based on tags, you provide tag information in the condition element of a policy using the `aws:ResourceTag/key-name`, `aws:RequestTag/key-name`, or `aws:TagKeys` condition keys.

If a service supports all three condition keys for every resource type, then the value is **Yes** for the service. If a service supports all three condition keys for only some resource types, then the value is **Partial**.

For more information about ABAC, see Define permissions with ABAC authorization in the *IAM User Guide*. To view a tutorial with steps for setting up ABAC, see Use attribute-based access control (ABAC) in the *IAM User Guide*.

Using temporary credentials with Amazon Cognito

Supports temporary credentials: Yes

Temporary credentials provide short-term access to AWS resources and are automatically created when you use federation or switch roles. AWS recommends that you dynamically generate temporary credentials instead of using long-term access keys. For more information, see Temporary security credentials in IAM and AWS services that work with IAM in the *IAM User Guide*.

Cross-service principal permissions for Amazon Cognito

Supports forward access sessions (FAS): No

Forward access sessions (FAS) use the permissions of the principal calling an AWS service, combined with the requesting AWS service to make requests to downstream services. For policy details when making FAS requests, see Forward access sessions.

Service roles for Amazon Cognito

Supports service roles: Yes

A service role is an IAM role that a service assumes to perform actions on your behalf. An IAM administrator can create, modify, and delete a service role from within IAM. For more information, see Create a role to delegate permissions to an AWS service in the *IAM User Guide*.

For details about Amazon Cognito service roles, see Activate push synchronization and Implementing push synchronization.

> **Warning**
>
> Changing the permissions for a service role might break Amazon Cognito functionality. Edit service roles only when Amazon Cognito provides guidance to do so.

Service-linked roles for Amazon Cognito

Supports service-linked roles: Yes

A service-linked role is a type of service role that is linked to an AWS service. The service can assume the role to perform an action on your behalf. Service-linked roles appear in your AWS account and are owned by the service. An IAM administrator can view, but not edit the permissions for service-linked roles.

For details about creating or managing Amazon Cognito service-linked roles, see Using service-linked roles for Amazon Cognito.

Identity-based policy examples for Amazon Cognito

By default, users and roles don't have permission to create or modify Amazon Cognito resources. To grant users permission to perform actions on the resources that they need, an IAM administrator can create IAM policies.

To learn how to create an IAM identity-based policy by using these example JSON policy documents, see Create IAM policies (console) in the *IAM User Guide*.

For details about actions and resource types defined by Amazon Cognito, including the format of the ARNs for each of the resource types, see Actions, resources, and condition keys for Amazon Cognito in the *Service Authorization Reference*.

Topics

- Policy best practices
- Using the Amazon Cognito console
- Allow users to view their own permissions
- Restricting console access to a specific identity pool
- Allowing access to specific dataset for all identities in a pool

Policy best practices

Identity-based policies determine whether someone can create, access, or delete Amazon Cognito resources in your account. These actions can incur costs for your AWS account. When you create or edit identity-based policies, follow these guidelines and recommendations:

- **Get started with AWS managed policies and move toward least-privilege permissions** – To get started granting permissions to your users and workloads, use the *AWS managed policies* that grant permissions for many common use cases. They are available in your AWS account. We recommend that you reduce permissions further by defining AWS customer managed policies that are specific to your use cases. For more information, see AWS managed policies or AWS managed policies for job functions in the *IAM User Guide*.
- **Apply least-privilege permissions** – When you set permissions with IAM policies, grant only the permissions required to perform a task. You do this by defining the actions that can be taken on specific resources under specific conditions, also known as *least-privilege permissions*. For more information about using IAM to apply permissions, see Policies and permissions in IAM in the *IAM User Guide*.
- **Use conditions in IAM policies to further restrict access** – You can add a condition to your policies to limit access to actions and resources. For example, you can write a policy condition to specify that all requests must be sent using SSL. You can also use conditions to grant access to service actions if they are used through a specific AWS service, such as CloudFormation. For more information, see IAM JSON policy elements: Condition in the *IAM User Guide*.
- **Use IAM Access Analyzer to validate your IAM policies to ensure secure and functional permissions** – IAM Access Analyzer validates new and existing policies so that the policies adhere to the IAM policy language (JSON) and IAM best practices. IAM Access Analyzer provides more than 100 policy checks and actionable recommendations to help you author secure and functional policies. For more information, see Validate policies with IAM Access Analyzer in the *IAM User Guide*.

- **Require multi-factor authentication (MFA)** – If you have a scenario that requires IAM users or a root user in your AWS account, turn on MFA for additional security. To require MFA when API operations are called, add MFA conditions to your policies. For more information, see Secure API access with MFA in the *IAM User Guide*.

For more information about best practices in IAM, see Security best practices in IAM in the *IAM User Guide*.

> **Note**
>
> The original and new versions of the Amazon Cognito console have different underlying behavior when you view and modify your Amazon Cognito resources. If you granted permission to actions under the `cognito-idp` service prefix only when the condition `aws:ViaAWSService` is true, the affected IAM principal could have been effective for Amazon Cognito resources in the original console, but not the new console. To work in the Amazon Cognito console, don't set an `aws:ViaAWSService` condition on Amazon Cognito permissions in your IAM policy.

Using the Amazon Cognito console

To access the Amazon Cognito console, you must have a minimum set of permissions. These permissions must allow you to list and view details about the Amazon Cognito resources in your AWS account. If you create an identity-based policy that is more restrictive than the minimum required permissions, the console won't function as intended for entities (users or roles) with that policy.

You don't need to allow minimum console permissions for users that are making calls only to the AWS CLI or the AWS API. Instead, allow access to only the actions that match the API operation that they're trying to perform.

To ensure that users and roles can still use the Amazon Cognito console, also attach the Amazon Cognito `ConsoleAccess` or `ReadOnly` AWS managed policy to the entities. For more information, see Adding permissions to a user in the *IAM User Guide*.

Allow users to view their own permissions

This example shows how you might create a policy that allows IAM users to view the inline and managed policies that are attached to their user identity. This policy includes permissions to complete this action on the console or programmatically using the AWS CLI or AWS API.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Sid": "ViewOwnUserInfo",
            "Effect": "Allow",
            "Action": [
                "iam:GetUserPolicy",
                "iam:ListGroupsForUser",
                "iam:ListAttachedUserPolicies",
                "iam:ListUserPolicies",
                "iam:GetUser"
            ],
            "Resource": ["arn:aws-eusc:iam::*:user/${aws:username}"]
        },
        {
            "Sid": "NavigateInConsole",
            "Effect": "Allow",
            "Action": [
                "iam:GetGroupPolicy",
                "iam:GetPolicyVersion",
                "iam:GetPolicy",
                "iam:ListAttachedGroupPolicies",
                "iam:ListGroupPolicies",
                "iam:ListPolicyVersions",
                "iam:ListPolicies",
                "iam:ListUsers"
            ],
            "Resource": "*"
        }
    ]
}
```

Restricting console access to a specific identity pool

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:ListIdentityPools"
            ],
            "Resource": "*"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cognito-identity:*"
            ],
            "Resource": "arn:aws-eusc:cognito-identity:eusc-de-
east-1:111122223333:identitypool/eusc-de-east-1:1a1a1a1a-ffff-1111-9999-12345678"
        },
        {
            "Effect": "Allow",
            "Action": [
                "cognito-sync:*"
            ],
            "Resource": "arn:aws-eusc:cognito-sync:eusc-de-
east-1:111122223333:identitypool/eusc-de-east-1:1a1a1a1a-ffff-1111-9999-12345678"
        }
    ]
}
```

Allowing access to specific dataset for all identities in a pool

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
```

```
        {
            "Effect": "Allow",
            "Action": [
                "cognito-sync:ListRecords",
                "cognito-sync:UpdateRecords"
            ],
            "Resource": "arn:aws-eusc:cognito-sync:eusc-de-
east-1:111122223333:identitypool/eusc-de-east-1:1a1a1a1a-ffff-1111-9999-12345678/
identity/*/dataset/UserProfile"
        }
    ]
}
```

Troubleshooting Amazon Cognito identity and access

Use the following information to help you diagnose and fix common issues that you might encounter when working with Amazon Cognito and IAM.

Topics

- I am not authorized to perform an action in Amazon Cognito
- I am not authorized to perform iam:PassRole
- I'm an administrator and want to allow others to access Amazon Cognito
- I want to allow people outside of my AWS account to access my Amazon Cognito resources

I am not authorized to perform an action in Amazon Cognito

If you receive an error that you're not authorized to perform an action, your policies must be updated to allow you to perform the action.

The following example error occurs when the `mateojackson` IAM user tries to use the console to view details about a fictional *`my-example-widget`* resource but doesn't have the fictional `cognito-identity:`*`GetWidget`* permissions.

```
User: arn:aws-eusc:iam::123456789012:user/mateojackson is not authorized to perform:
 cognito-identity:GetWidget on resource: my-example-widget
```

In this case, the policy for the `mateojackson` user must be updated to allow access to the *`my-example-widget`* resource by using the `cognito-identity:`*`GetWidget`* action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I am not authorized to perform iam:PassRole

If you receive an error that you're not authorized to perform the `iam:PassRole` action, your policies must be updated to allow you to pass a role to Amazon Cognito.

Some AWS services allow you to pass an existing role to that service instead of creating a new service role or service-linked role. To do this, you must have permissions to pass the role to the service.

The following example error occurs when an IAM user named `marymajor` tries to use the console to perform an action in Amazon Cognito. However, the action requires the service to have permissions that are granted by a service role. Mary does not have permissions to pass the role to the service.

```
User: arn:aws-eusc:iam::123456789012:user/marymajor is not authorized to perform:
 iam:PassRole
```

In this case, Mary's policies must be updated to allow her to perform the `iam:PassRole` action.

If you need help, contact your AWS administrator. Your administrator is the person who provided you with your sign-in credentials.

I'm an administrator and want to allow others to access Amazon Cognito

To allow others to access Amazon Cognito, you must grant permission to the people or applications that need access. If you are using AWS IAM Identity Center to manage people and applications, you assign permission sets to users or groups to define their level of access. Permission sets automatically create and assign IAM policies to IAM roles that are associated with the person or application. For more information, see Permission sets in the *AWS IAM Identity Center User Guide*.

If you are not using IAM Identity Center, you must create IAM entities (users or roles) for the people or applications that need access. You must then attach a policy to the entity that grants them the correct permissions in Amazon Cognito. After the permissions are granted, provide the credentials

to the user or application developer. They will use those credentials to access AWS. To learn more about creating IAM users, groups, policies, and permissions, see IAM Identities and Policies and permissions in IAM in the *IAM User Guide*.

I want to allow people outside of my AWS account to access my Amazon Cognito resources

You can create a role that users in other accounts or people outside of your organization can use to access your resources. You can specify who is trusted to assume the role. For services that support resource-based policies or access control lists (ACLs), you can use those policies to grant people access to your resources.

To learn more, consult the following:

- To learn whether Amazon Cognito supports these features, see How Amazon Cognito works with IAM.
- To learn how to provide access to your resources across AWS accounts that you own, see Providing access to an IAM user in another AWS account that you own in the *IAM User Guide*.
- To learn how to provide access to your resources to third-party AWS accounts, see Providing access to AWS accounts owned by third parties in the *IAM User Guide*.
- To learn how to provide access through identity federation, see Providing access to externally authenticated users (identity federation) in the *IAM User Guide*.
- To learn the difference between using roles and resource-based policies for cross-account access, see Cross account resource access in IAM in the *IAM User Guide*.

Using service-linked roles for Amazon Cognito

Amazon Cognito uses AWS Identity and Access Management (IAM) service-linked roles. A service-linked role is a unique type of IAM role with a trust policy that permits an AWS service to assume the role. Service-linked roles are predefined by Amazon Cognito and include all the permissions that the service requires to call other AWS services on your behalf.

A service-linked role makes setting up Amazon Cognito easier because you don't have to manually add the necessary permissions. Amazon Cognito defines the permissions of its service-linked roles, and unless defined otherwise, only Amazon Cognito can assume its roles. The defined permissions include the trust policy and the permissions policy, and that permissions policy cannot be attached to any other IAM entity.

You can delete a service-linked role only after first deleting their related resources. This protects your Amazon Cognito resources because you can't inadvertently remove permission to access the resources.

For information about other services that support service-linked roles, see AWS Services That Work with IAM and look for the services that have **Yes** in the **Service-Linked Role** column. Choose a **Yes** with a link to view the service-linked role documentation for that service.

Service-linked role permissions for Amazon Cognito

Amazon Cognito uses the following service-linked roles:

- **AWSServiceRoleForAmazonCognitoIdpEmailService** – Allows Amazon Cognito user pools service to use your Amazon SES identities for sending email.

- **AWSServiceRoleForAmazonCognitoIdp** – Allows Amazon Cognito user pools to publish events and configure endpoints for your Amazon Pinpoint projects.

AWSServiceRoleForAmazonCognitoIdpEmailService

The `AWSServiceRoleForAmazonCognitoIdpEmailService` service-linked role trusts the following services to assume the role:

- `email.cognito-idp.amazonaws.com`

The role permissions policy allows Amazon Cognito to complete the following actions on the specified resources:

Allowed Actions for AWSServiceRoleForAmazonCognitoIdpEmailService:

- Action: `ses:SendEmail` and `ses:SendRawEmail`
- Resource: *

The policy denies Amazon Cognito the ability to complete the following actions on the specified resources:

Denied Actions

- Action: `ses:List*`

- Resource: *

With these permissions, Amazon Cognito can use your verified email addresses in Amazon SES only to email your users. Amazon Cognito emails your users when they perform certain actions in the client app for a user pool, such as signing up or resetting a password.

You must configure permissions to allow an IAM entity (such as a user, group, or role) to create, edit, or delete a service-linked role. For more information, see Service-linked role permissions in the *IAM User Guide*.

AWSServiceRoleForAmazonCognitoIdp

The AWSServiceRoleForAmazonCognitoIdp service-linked role trusts the following services to assume the role:

- `email.cognito-idp.amazonaws.com`

The role permissions policy allows Amazon Cognito to complete the following actions on the specified resources:

Allowed Actions for AWSServiceRoleForAmazonCognitoIdp

- Action: `cognito-idp:Describe`
- Resource: *

With this permission, Amazon Cognito can call `Describe` Amazon Cognito API operations for you.

> **Note**
>
> When you integrate Amazon Cognito with Amazon Pinpoint using `createUserPoolClient` and `updateUserPoolClient`, resource permissions will be added to the SLR as an inline policy. The inline policy will provide `mobiletargeting:UpdateEndpoint` and `mobiletargeting:PutEvents` permissions. These permissions allow Amazon Cognito to publish events and configure endpoints for Pinpoint projects you integrate with Cognito.

Creating a service-linked role for Amazon Cognito

You don't need to manually create a service-linked role. When you configure a user pool to use your Amazon SES configuration to handle email delivery in the AWS Management Console, the AWS CLI, or the Amazon Cognito API, Amazon Cognito creates the service-linked role for you.

If you delete this service-linked role, and then need to create it again, you can use the same process to recreate the role in your account. When you configure a user pool to use your Amazon SES configuration to handle email delivery, Amazon Cognito creates the service-linked role for you again.

Before Amazon Cognito can create this role, the IAM permissions that you use to set up your user pool must include the `iam:CreateServiceLinkedRole` action. For more information about updating permissions in IAM, see Changing Permissions for an IAM User in the *IAM User Guide*.

Editing a service-linked role for Amazon Cognito

You can't edit the AmazonCognitoIdp or AmazonCognitoIdpEmailService service-linked roles in AWS Identity and Access Management. After you create a service-linked role, you can't change the name of the role because various entities might reference the role. However, you can edit the description of the role using IAM. For more information, see Editing a service-linked role in the *IAM User Guide*.

Deleting a service-linked role for Amazon Cognito

If you no longer need to use a feature or service that requires a service-linked role, we recommend that you delete that role. If you delete the role, you only retain entities that Amazon Cognito actively monitors or maintains. Before you can delete AmazonCognitoIdp or AmazonCognitoIdpEmailService service-linked roles, you must do one of the following for each user pool that uses the role:

- Delete the user pool.
- Update the email settings in the user pool to use the default email functionality. The default setting doesn't use the service-linked role.

Remember to perform the action in each AWS Region with a user pool that uses the role.

> **Note**
>
> If the Amazon Cognito service is using the role when you try to delete the resources, then the deletion might fail. If that happens, wait for a few minutes and try the operation again.

To delete an Amazon Cognito user pool

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to delete.
4. Choose **Delete pool**.
5. In the **Delete user pool** window, type **`delete`**, and choose **Delete pool**.

To update an Amazon Cognito user pool to use the default email functionality

1. Sign in to the AWS Management Console and open the Amazon Cognito console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cognito.
2. Choose **Manage User Pools**.
3. On the **Your User Pools** page, choose the user pool that you want to update.
4. In the navigation menu on the left, choose **Message customizations**.
5. Under **Do you want to send emails through your Amazon SES Configuration?**, choose **No - Use Cognito (Default)**.
6. When you finish setting your email account options, choose **Save changes**.

To manually delete the service-linked role using IAM

Use the IAM console, the AWS CLI, or the AWS API to delete AmazonCognitoIdp or AmazonCognitoIdpEmailService service-linked roles. For more information, see Deleting a service-linked role in the *IAM User Guide*.

Supported Regions for Amazon Cognito service-linked roles

Amazon Cognito supports service-linked roles in all AWS Regions where the service is available. For more information, see AWS Regions and Endpoints.

Logging and monitoring in Amazon Cognito

Monitoring is an important part of maintaining the reliability, availability, and performance of Amazon Cognito and your other AWS solutions. Amazon Cognito currently supports the following AWS services so that you can monitor your organization and the activity that happens within it.

- **AWS CloudTrail –** With CloudTrail you can capture API calls from the Amazon Cognito console and from code calls to the Amazon Cognito API operations. For example, when a user authenticates, CloudTrail can record details such as the IP address in the request, who made the request, and when it was made.
- **Amazon CloudWatch Logs** – With CloudWatch Logs, you can send fine-grained logs of user activity to a log group. For example, you can review detailed user activity logs to troubleshoot the delivery of email and SMS messages to your users.
- **Amazon CloudWatch Metrics –** With CloudWatch metrics you can monitor, report, and take automatic actions in case of an event in near real time. For example, you can create CloudWatch dashboards on the provided metrics to monitor your Amazon Cognito user pools, or you can create CloudWatch alarms on the provided metrics to notify you on breach of a set threshold.
- **Amazon CloudWatch Logs Insights** – With CloudWatch Logs Insights, you can configure CloudTrail to send events to CloudWatch for monitoring Amazon Cognito CloudTrail log files.

Topics

- Monitoring and managing costs
- Exporting logs from Amazon Cognito user pools
- Tracking quotas and usage in CloudWatch and Service Quotas
- Amazon Cognito logging in AWS CloudTrail

Monitoring and managing costs

Like with any other AWS service, it's important to understand the effect of your Amazon Cognito configuration and usage on your AWS bill. As part of your preparations for the deployment of user pools to production, set up monitoring and safeguards for activity and resource consumption. When you know where to look and what actions produce additional cost, you can set up precautions against surprises in your bill.

Amazon Cognito charges for the following dimensions of your usage.

- User pool monthly active users (MAUs)—rate varies by feature plan
- User pool MAUs signed in with OIDC or SAML federation
- Request volume for machine to machine (M2M) authorization, from both `client_credentials` grants at the token endpoint and the `GetClientToken` API operation
- Purchased usage above default quotas for some categories of user pool APIs

Additionally, features of your user pool like email messages, SMS messages, and Lambda triggers can incur costs in dependent services. For a complete overview, see Amazon Cognito Pricing.

Viewing and anticipating costs

High-volume events like product launches and opening up to new userbases can increase your MAU count and have a cost impact. Estimate the new user count in advance and watch activity as it happens. You might find that you want to accommodate the volume with a purchase of additional quota capacity, or control the volume with additional security measures.

You can view and report on your AWS costs in the AWS Billing and Cost Management console. You can find your most recent charges for Amazon Cognito in the **Billing and payments** section. Under **Bills**, **Charges by service**, filter on `Cognito` to view your usage. For more information, see Viewing your bill in the *AWS Billing User Guide*.

To monitor API request rates, review the **Utilization** metric in the Service Quotas console. For example, both `client_credentials` grant requests to the token endpoint and GetClientToken API requests display as **Rate of ClientAuthentication requests**, because they share a request quota. In your bill, these requests are associated with the app client that produced them. With this information, you can equitably allocate costs to the tenants in a multi-tenant architecture.

To get a count of token-endpoint M2M requests for a period of time, you can also send AWS CloudTrail events to CloudWatch Logs for analysis. Query your CloudTrail events for `Token_POST` events with a client credentials grant. The following CloudWatch Insights query returns this count. To include M2M requests that you make with the `GetClientToken` API operation, use the **Rate of ClientAuthentication requests** utilization metric described earlier in this section.

```
filter eventName = "Token_POST" and @message like '"grant_type":["client_credentials"]'
 | stats count(*)
```

Managing costs

Amazon Cognito bills based on user count, feature usage, and request volume. The following are some tips to manage cost in Amazon Cognito,

Don't activate inactive users

Typical operations that make a user active are sign-in, sign-up, and password reset. For a more thorough list, see Monthly active users. Amazon Cognito doesn't count inactive users toward your bill. Avoid any operations that set a user active. Instead of the AdminGetUser API operation, query users with the ListUsers operation. Don't perform high-volume administrative testing of user pool operations with inactive users.

Link federated users

Users who sign in with a SAML 2.0 or OpenID Connect (OIDC) identity provider have a higher cost than local users. You can link these users to a local user profile. A linked user can sign in as a local user with the attributes and access that come with their federated user. Users from SAML or OIDC IdPs who, in the course of a month, only sign in with a linked local account are billed as local users.

Manage request rates

If your user pool is approaching the upper limit of your quota, you might consider purchasing additional capacity to handle the volume. You might be able to reduce the volume of requests in your application. For more information, see Optimize request rates for quota limits.

Request a new token only when you need one

Machine to machine (M2M) authorization with client credentials grants can reach a high volume of token requests. Each new token request has an effect on your request-rate quota and the size of your bill. To optimize cost, include token expiration settings and token handling in the design of your applications.

- Cache access tokens so that when your application requests a new token, it receives a cached version of a previously-issued token. When you implement this method, your caching proxy acts as a guard against applications that request access tokens without awareness of the expiration of previously-acquired tokens. Caching tokens is ideal for short-lived microservices like Lambda functions and Docker containers.
- Implement token-handling mechanisms in your applications that account for token expiration. Don’t request a new token until previous tokens are about to expire. As a best practice, refresh

tokens at about 75% of the token lifetime. This practice maximizes token duration while ensuring user continuity in your application.

Evaluate the confidentiality and availability needs of each application and configure the user pool app client to issue access tokens with an appropriate validity period. Custom token duration works best with longer-lived APIs and servers that can persistently manage the frequency of requests for credentials.

ListUsers, not AdminGetUser

To query the attributes of users in your user pool, use the ListUsers API operation and associated SDK methods when possible. AdminGetUser marks a user as active for the month and contributes to the monthly active users (MAUs) that are used to calculate your bill for user pools.

Manage feature plans

When you choose a feature plan in a user pool, the billing rate applies to all MAUs in the user pool. If you have users that don't need features that come with a higher-level feature plan, separate them into another user pool.

Exporting logs from Amazon Cognito user pools

You can configure your user pool to send detailed logs of some additional activity to another AWS service, like a CloudWatch log group. These logs are of a finer granularity than those in AWS CloudTrail, and can be useful to troubleshoot your user pool and analyze user sign-in activity with threat protection. When you want to stream logs of SMS and email notification errors, your user pool sends `ERROR`-level logs to a CloudWatch log group. When you want to stream logs of user sign-in activity, your user pool sends `INFO`-level logs to a log group, a Amazon Data Firehose stream, or an Amazon S3 bucket. You can combine both options in a user pool.

Topics

- Things to know about log export
- Exporting email and SMS message delivery errors
- Exporting threat protection user activity logs

Things to know about log export

Cost impact

Amazon Data Firehose, Amazon S3, and CloudWatch Logs incur costs for data ingestion and retrieval. Your logging configuration might affect your AWS bill. For more information, see the following:

- Vended Logs in *Amazon CloudWatch Pricing*.
- Amazon Data Firehose pricing
- Amazon S3 pricing

User-activity log exports contain security assessments and are a function of user pool threat protection. Amazon Cognito only generates these logs when threat protection is in **Audit-only** or **Full-function** mode and your user pool is on the Plus feature plan.

User activity logs are `INFO` level

Exported user-activity logs are at the `INFO` error level only and provide information for statistical and security analysis of authentication activity. Messages at the `WARNING` and `ERROR` error levels, for example throttling errors, aren't included in the exported logs.

Best-effort delivery

Delivery of logs from Amazon Cognito is best effort. The volume of logs that your user pool delivers, and your service quotas for CloudWatch Logs, Amazon S3, and Firehose can affect the delivery of logs.

Existing external logs are unaffected

These logging options don't replace or change the following log functions of user pools.

1. CloudTrail logs of routine user activity like sign-up and sign-in.
2. Analysis of user activity at scale with CloudWatch metrics.

Separately, you can also find logs from Viewing the user pool import results in the CloudWatch console and Customizing user pool workflows with Lambda triggers in CloudWatch Logs. Amazon Cognito and Lambda store these logs in different log groups from the ones that you specify for user activity logs.

Applies only to user pools

No log export capabilities exist for identity pools.

Requires user permissions and service-linked role

The AWS principal that sets up log export must have permissions to modify the target resources, as described in the topics that follow. Amazon Cognito creates a service-linked role on your behalf and assumes the role to deliver logs to the target resource.

For more information about the authorization model for sending logs from Amazon Cognito, see Enable logging from AWS services in the *Amazon CloudWatch Logs User Guide*.

Log level is exclusive to log type

Message-delivery logs are of the `userNotification` type and of the `ERROR` errorlevel. Advanced security user activity logs are of the `userAuthEvents` type and of the `INFO` errorlevel. You can combine two members of `LogConfigurations`, one for `userNotification` to CloudWatch Logs, and one for `userAuthEvents` to Firehose, Amazon S3, or CloudWatch Logs.

You can't send user-activity logs to multiple destinations. You can't send user-notification logs to any destination other than CloudWatch Logs.

Different configuration options

You can only configure user-notification logs with the Amazon Cognito user pools API or an AWS SDK. You can configure advanced security user-activity logs with the API or in the Amazon Cognito console. To set both, use the API as demonstrated in the example request at SetLogDeliveryConfiguration.

Additional configuration required with large resource-based policies

To send logs to log groups with a resource policy of a size greater than 5120 characters, configure a log group with a path that starts with `/aws/vendedlogs`. For more information, see Enabling logging from certain AWS services.

Automatic creation of a folder in Amazon S3

When you configure threat protection log export to an Amazon S3 bucket, Amazon Cognito might create an `AWSLogs` folder in your bucket. That folder is not created in all cases, and the configuration can succeed without creating it.

Exporting email and SMS message delivery errors

For email and SMS message delivery errors, you can deliver **Error**-level user notification logs from your user pool. When you activate this feature, you can choose the log group where you want

Amazon Cognito to send logs. User notification logging is useful when you want to find out the status of email and SMS messages that your user pool delivered with Amazon SNS and Amazon SES. This log export option, unlike user-activity export, doesn't require the Plus feature plan.

You can configure detailed notification logs with the Amazon Cognito user pools API in a SetLogDeliveryConfiguration API request. You can view the logging configuration of a user pool in a GetLogDeliveryConfiguration API request. The following is an example request body.

```
{
   "LogConfigurations": [
      {
         "CloudWatchLogsConfiguration": {
            "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:example-user-
pool-exported"
         },
         "EventSource": "userNotification",
         "LogLevel": "ERROR"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

You must authorize these requests with AWS credentials that have the following permissions.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
            {
            "Sid": "ManageUserPoolLogs",
            "Action": [
                "cognito-idp:SetLogDeliveryConfiguration",
                "cognito-idp:GetLogDeliveryConfiguration"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
```

```
            "Sid": "CognitoLog",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "CognitoLoggingCWL",
            "Action": [
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

The following is an example event from a user pool. This log schema is subject to change. Some fields might be logged with null values.

```
{
    "eventTimestamp": "1687297330677",
    "eventSource": "USER_NOTIFICATION",
    "logLevel": "ERROR",
    "message": {
        "details": "String"
    },
    "logSourceId": {
        "userPoolId": "String"
    }
}
```

Exporting threat protection user activity logs

User pools with the Plus feature plan and threat protection log user activity events: the details and security assessment of user sign-in, sign-out, and other authentication operations with your user pool. You might want to review user activity logs in your own log-management system, or create an archive. You can export this data to a Amazon CloudWatch Logs log group, an Amazon Data Firehose stream, or an Amazon Simple Storage Service (Amazon S3) bucket. From there, you can ingest this data into other systems that analyze, normalize or otherwise process data in ways that fit it in to your operational processes. To export data of this type, your user pool must be on the Plus feature plan and threat protection must be active in your user pool.

With the information in these user activity logs, you can view a profile of user sign-in and account-management activity. By default, Amazon Cognito captures these events to storage that's based in your user pool. The following example is an example event for a user who signed in and was evaluated to have no risk factors. You can retrieve this information with the `AdminListUserAuthEvents` API operation. The following is an example output:


```
{
    "AuthEvents": [
        {
            "EventId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
            "EventType": "SignIn",
            "CreationDate": "2024-06-27T10:49:59.139000-07:00",
            "EventResponse": "Pass",
            "EventRisk": {
                "RiskDecision": "NoRisk",
                "CompromisedCredentialsDetected": false
            },
            "ChallengeResponses": [
                {
                    "ChallengeName": "Password",
                    "ChallengeResponse": "Success"
                }
            ],
            "EventContextData": {
                "IpAddress": "192.0.2.1",
                "DeviceName": "Chrome 126, Windows 10",
                "Timezone": "-07:00",
                "City": "null",
                "Country": "United States"
            }
```

```
        }
    ],
    "NextToken": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222#2024-06-27T17:49:59.139Z"
}
```

You can activate log export for user activity in the Amazon Cognito console or with the SetLogDeliveryConfiguration API operation.

AWS Management Console

1. If you don't already have one that you want to use, create an S3 bucket, Firehose stream, or CloudWatch log group.
2. Sign in to the Amazon Cognito console.
3. Choose **User Pools**.
4. Choose an existing user pool from the list, or create a user pool.
5. Choose the **Advanced security** tab. Locate **Export user activity logs** and choose **Edit**
6. Under **Logging status**, select the checkbox next to **Activate user activity log export**.
7. Under **Logging destination**, choose the AWS service that you want to handle your logs: **CloudWatch log group**, **Amazon Data Firehose stream**, or **S3 bucket**.
8. Your selection will populate the resource selector with the corresponding resource type. Select a log group, stream, or bucket from the list. You can also select the **Create** button to navigate to the AWS Management Console for the selected service and create a new resource.
9. Select **Save changes**.

API

Choose one type of destination for your user activity logs.

The following is an example `SetLogDeliveryConfiguration` request body that sets a Firehose stream as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "FirehoseConfiguration": {
```

```
            "StreamArn": "arn:aws:firehose:us-west-2:123456789012:deliverystream/
example-user-pool-activity-exported"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

The following is an example `SetLogDeliveryConfiguration` request body that sets a Amazon S3 bucket as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "S3Configuration": {
            "BucketArn": "arn:aws:s3:::amzn-s3-demo-logging-bucket"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

The following is an example `SetLogDeliveryConfiguration` request body that sets a CloudWatch log group as the log destination.

```
{
   "LogConfigurations": [
      {
         "EventSource": "userAuthEvents",
         "CloudWatchLogsConfiguration": {
            "LogGroupArn": "arn:aws:logs:us-west-2:123456789012:log-group:DOC-
EXAMPLE-LOG-GROUP"
         },
         "LogLevel": "INFO"
      }
   ],
   "UserPoolId": "us-west-2_EXAMPLE"
}
```

The user that configures log delivery must be a user pool administrator and have the following additional permissions:

Amazon S3

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageUserPoolLogs",
            "Action": [
                "cognito-idp:SetLogDeliveryConfiguration",
                "cognito-idp:GetLogDeliveryConfiguration"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "ManageLogsS3",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "s3:PutBucketPolicy",
                "s3:GetBucketPolicy"
            ],
            "Resource": "*"
        }
    ]
}
```

CloudWatch Logs

JSON

```
{
    "Version":"2012-10-17",
```

```
    "Statement": [
        {
            "Sid": "ManageUserPoolLogs",
            "Action": [
                "cognito-idp:SetLogDeliveryConfiguration",
                "cognito-idp:GetLogDeliveryConfiguration"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "ManageLogsCWL",
            "Action": [
                "logs:CreateLogDelivery",
                "logs:GetLogDelivery",
                "logs:UpdateLogDelivery",
                "logs:DeleteLogDelivery",
                "logs:ListLogDeliveries",
                "logs:PutResourcePolicy",
                "logs:DescribeResourcePolicies",
                "logs:DescribeLogGroups"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        }
    ]
}
```

Amazon Data Firehose

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Sid": "ManageUserPoolLogs",
            "Action": [
```

```
                "cognito-idp:SetLogDeliveryConfiguration",
                "cognito-idp:GetLogDeliveryConfiguration"
            ],
            "Resource": [
                "*"
            ],
            "Effect": "Allow"
        },
        {
            "Sid": "ManageUserPoolLogsFirehose",
            "Effect": "Allow",
            "Action": [
                "logs:CreateLogDelivery",
                "iam:CreateServiceLinkedRole",
                "firehose:TagDeliveryStream"
            ],
            "Resource": "*"
        }
    ]
}
```

The following is an example event from a user pool. This log schema is subject to change. Some fields might be logged with null values.

```
{
    "eventTimestamp": "1687297330677",
    "eventSource": "USER_ACTIVITY",
    "logLevel": "INFO",
    "message": {
        "version": "1",
        "eventId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE11111",
        "eventType": "SignUp",
        "userSub": "a1b2c3d4-5678-90ab-cdef-EXAMPLE22222",
        "userName": "test-user",
        "userPoolId": "us-west-2_EXAMPLE",
        "clientId": "1example23456789",
        "creationDate": "Wed Jul 17 17:25:55 UTC 2024",
        "eventResponse": "InProgress",
        "riskLevel": "",
        "riskDecision": "PASS",
        "challenges": [],
        "deviceName": "Other, Other",
```

```
        "ipAddress": "192.0.2.1",
        "requestId": "a1b2c3d4-5678-90ab-cdef-EXAMPLE33333",
        "idpName": "",
        "compromisedCredentialDetected": "false",
        "city": "Seattle",
        "country": "United States",
        "eventFeedbackValue": "",
        "eventFeedbackDate": "",
        "eventFeedbackProvider": "",
        "hasContextData": "true"
    },
    "logSourceId": {
        "userPoolId": "us-west-2_EXAMPLE"
    }
}
```

Tracking quotas and usage in CloudWatch and Service Quotas

You can monitor Amazon Cognito user pools using Amazon CloudWatch or using Service Quotas. You can also monitor identity pools usage in Service Quotas. CloudWatch collects raw data and processes it into readable, near real-time metrics. In CloudWatch, you can set alarms that watch for certain thresholds and send notifications or take actions when those thresholds are met. To create a CloudWatch alarm for a service quota, see Create a CloudWatch alarm. Amazon Cognito metrics are available at five minute intervals. For more information about retention periods in CloudWatch, visit the Amazon CloudWatch FAQ page.

You can use Service Quotas to view and manage your Amazon Cognito user pools and identity pools quota usage. The Service Quotas console has three features: view service quotas, request a service quota increase, and view current utilization. You can use the first feature to view quotas and see whether the quota is adjustable. You can use the second feature to request a Service Quotas increase. You can use the last feature to view quota utilization. This feature is only available after your account has been active for a while. For more information on viewing quotas in the Service Quotas console, see Viewing Service Quotas.

> **Note**
>
> Amazon Cognito metrics are available at 5 minute intervals. For more information about retention periods in CloudWatch, visit the Amazon CloudWatch FAQ page.

If you are signed in to an AWS account that is set up as a monitoring account in CloudWatch cross-account observability, you can use that monitoring account to visualize service quotas and set alarms for metrics in the source accounts that are linked to that monitoring account. For more information, see CloudWatch cross-account observability.

Topics

- User pool metrics in CloudWatch
- Metrics in Service Quotas

User pool metrics in CloudWatch

User pools report user-activity statistics to CloudWatch as metrics. From CloudWatch, you can analyze the volume of authentication activity and quota usage in your user pools. With the information in these metrics, you can set alarms for noteworthy events and adjust your user pool configuration as needed. Where user-activity logging has detailed records of user activity in your user pools, CloudWatch metrics have aggregated statistics and performance indicators.

The following table lists the metrics available for Amazon Cognito user pools. Amazon Cognito publishes metrics to the namespaces `AWS/Cognito` and `AWS/Usage`. For more information, see Namespaces in *Amazon CloudWatch User Guide*.

For more information about tracking quotas and usage, see Track quota usage and Track monthly active users (MAUs).

> **Note**
>
> Metrics that haven't had any new data points in the past two weeks don't appear in the console. They also don't appear when you enter their metric name or dimension names in the search box in the **All metrics** tab in the console. In addition, they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the `get-metric-data` or `get-metric-statistics` commands in the AWS CLI.

Metric	Description	Namespace
`SignUpSuccesses`	Provides the total number of successful user registrat	`AWS/Cognito`

Metric	Description	Namespace
	ion requests made to the Amazon Cognito user pool. A successful user registration request produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful user registration requests, use the `Average` statistic on this metric. To count the total number of user registration requests, use the `Sample Count` statistic on this metric. To count the total number of successful user registration requests, use the `Sum` statistic on this metric. To count the total number of failed user registration requests, use the CloudWatch `Math` expression and subtract the `Sum` statistic from the `Sample Count` statistic. This metric is published for each user pool for each user pool client. In case when the user registration is performed	

Metric	Description	Namespace
	by an admin, the metric is published with the user pool client as `Admin`. Note that this metric is not emitted for User import and User migration cases. Metric dimension: `UserPool`, `UserPoolClient` Units: Count	

Metric	Description	Namespace
SignUpThrottles	Provides the total number of throttled user registration requests made to the Amazon Cognito user pool. A count of 1 is published whenever a user registration request is throttled. To count the total number of throttled user registration requests, use the Sum statistic for this metric. This metric is published for each user pool for each client. In case when the request that was throttled was made by an administrator, the metric is published with user pool client as Admin. Metric dimension: UserPool, UserPoolClient Units: Count	AWS/Cognito

Metric	Description	Namespace
SignInSuccesses	Provides the total number of successful user authentication requests made to the Amazon Cognito user pool. A user authentication is considered successful when authentication token is issued to the user. A successful authentication produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful user authentication requests, use the Average statistic on this metric. To count the total number of user authentication requests, use the Sample Count statistic on this metric. To count the total number of successful user authentication requests, use the Sum statistic on this metric. To count the total number of failed user authentication requests, use the CloudWatch Math expression and subtract the Sum statistic from the Sample Count statistic.	AWS/Cognito

Metric	Description	Namespace
	This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, the correspon ding user pool client value in the metric contains a fixed value `Invalid` instead of the actual invalid value sent in the request. Note that requests to refresh the Amazon Cognito token is not included in this metric. There is a separate metric for providing `Refresh` token statistics. Metric dimension: `UserPool`, `UserPoolClient` Units: Count	

Metric	Description	Namespace
SignInThrottles	Provides the total number of throttled user authentication requests made to the Amazon Cognito user pool. A count of 1 is published whenever an authentication request is throttled. To count the total number of throttled user authentication requests, use the Sum statistic for this metric. This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, the correspon ding user pool client value in the metric contains a fixed value Invalid instead of the actual invalid value sent in the request. Requests to refresh Amazon Cognito token is not included in this metric. There is a separate metric for providing Refresh token statistics. Metric dimension: UserPool, UserPoolClient Units: Count	AWS/Cognito

Metric	Description	Namespace
TokenRefreshSuccesses	Provides the total number of successful requests to refresh an Amazon Cognito token that were made to the Amazon Cognito user pool. A successful refresh Amazon Cognito token request produces a value of 1, whereas an unsuccessful request produces a value of 0. A throttled request is also considered as an unsuccessful request, and hence a throttled request will also produce a count of 0. To find the percentage of successful requests to refresh an Amazon Cognito token, use the Average statistic on this metric. To count the total number of requests to refresh an Amazon Cognito token, use the Sample Count statistic on this metric. To count the total number of successful requests to refresh an Amazon Cognito token, use the Sum statistic on this metric. To count the total number of failed requests to refresh an Amazon Cognito token, use the CloudWatch Math expression and subtract	AWS/Cognito

Metric	Description	Namespace
	the Sum statistic from the Sample Count statistic. This metric is published per each user pool client. If an invalid user pool client is in a request, the user pool client value contains a fixed value of Invalid. Metric dimension: UserPool, UserPoolClient Units: Count	

Metric	Description	Namespace
TokenRefreshThrottles	Provides the total number of throttled requests to refresh an Amazon Cognito token that were made to the Amazon Cognito user pool. A count of 1 is published whenever a refresh Amazon Cognito token request is throttled. To count the total number of throttled requests to refresh an Amazon Cognito token, use the Sum statistic for this metric. This metric is published for each user pool for each client. In case an invalid user pool client is provided with a request, correspon ding user pool client value in the metric contains a fixed value Invalid instead of the actual invalid value sent in the request. Metric dimension: UserPool, UserPoolClient Units: Count	AWS/Cognito

Metric	Description	Namespace
FederationSuccesses	Provides the total number of successful identity federation requests to the Amazon Cognito user pool. An identity federation is considered successful when Amazon Cognito issues authentication tokens to the user. A successful identity federation request produces a value of 1, whereas an unsuccessful request produces a value of 0. Throttled requests and requests that generate an authorization code but no tokens produce a value of 0. To find the percentage of successful identity federation requests, use the Average statistic on this metric. To count the total number of identity federation requests, use the Sample Count statistic on this metric. To count the total number of successful identity federation requests, use the Sum statistic on this metric. To count the total number of failed identity federation requests, use the CloudWatch Math expression and subtract the Sum statistic from the Sample Count statistic.	AWS/Cognito

Metric	Description	Namespace
	Metric dimension: `UserPool`, `UserPoolClient` , `IdentityProvider` Units: Count	
`FederationThrottles`	Provides the total number of throttled identity federatio n requests to the Amazon Cognito user pool. A count of 1 is published whenever an identity federation request is throttled. To count the total number of throttled identity federation requests, use the `Sum` statistic for this metric. Metric dimension: `UserPool`, `UserPoolClient` , `IdentityProvider` Units: Count	`AWS/Cognito`

Metric	Description	Namespace
`CallCount`	Provides the total number of calls customers made related to a category. This metric includes all the calls, such as throttled calls, failed calls, and successful calls. The category quota is enforced for each AWS account across all user pools in an account and Region. You can count the total number of calls in a category using the `Sum` statistic for this metric. Metric dimension: Service, Type, Resource, Class Units: Count	`AWS/Usage`

Metric	Description	Namespace
`ThrottleCount`	Provides the total number of throttled calls related to a category. This metric is published at the account level. You can count the total number of calls in a category, using the `Sum` statistic for this metric. Metric dimension: Service, Type, Resource, Class Units: Count	`AWS/Usage`

Viewing threat protection metrics

The metrics that your user pool publishes have statistical information about the effect that your threat protection settings have on user authentication activity. You might want to know how many users are attempting to sign in with compromised credentials. You can also find out what percentage of sign-in activity was evaluated to have some level of risk. Amazon Cognito publishes metrics for threat protection features to your account in Amazon CloudWatch. Amazon Cognito groups the threat protection metrics together by risk level and also by request level.

To add context to your risk analysis, you can view information about individual user sign-in attempts, either in your user pool or in an exported data source.

To view metrics in the CloudWatch console

1. Open the CloudWatch console at https://eusc-de-east-1.console.amazonaws-eusc.eu/cloudwatch/.
2. In the navigation pane, choose **Metrics**.
3. Choose Amazon Cognito.
4. Choose a group of aggregated metrics, such as **By Risk Classification**.

5. The **All metrics** tab displays all metrics for that choice. You can do the following:

   - To sort the table, use the column heading.
   - To graph a metric, select the check box next to the metric. To select all metrics, select the check box in the heading row of the table.
   - To filter by resource, choose the resource ID, and then choose **Add to search**.
   - To filter by metric, choose the metric name, and then choose **Add to search**.

Metric	Description	Metric Dimensions	Namespace
CompromisedCredentialRisk	Requests where Amazon Cognito detected compromised credentials.	Operation: The type of operation. `PasswordChange`, `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool. RiskLevel: high (default), medium, or low.	`AWS/Cognito`
AccountTakeoverRisk	Requests where Amazon Cognito detected account take-over risk.	Operation: The type of operation. `PasswordChange`, `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool.	`AWS/Cognito`

Metric	Description	Metric Dimensions	Namespace
		RiskLevel: high, medium, or low.	
OverrideBlock	Requests that Amazon Cognito blocked because of the configuration provided by the developer.	Operation: The type of operation . `PasswordChange` , `SignIn`, or `SignUp` are the only dimensions. UserPoolId: The identifier of the user pool. RiskLevel: high, medium, or low.	`AWS/Cognito`
Risk	Requests that Amazon Cognito marked as risky.	Operation: The type of operation, such as `PasswordChange` , `SignIn`, or `SignUp`. UserPoolId: The identifier of the user pool.	`AWS/Cognito`
NoRisk	Requests where Amazon Cognito did not identify any risk.	Operation: The type of operation, such as `PasswordChange` , `SignIn`, or `SignUp`. UserPoolId: The identifier of the user pool.	`AWS/Cognito`

Amazon Cognito offers you two predefined groups of metrics for ready analysis in CloudWatch. **By Risk Classification** identifies the granularity of the risk level for requests that Amazon Cognito identifies as risky. **By Request Classification** reflects metrics aggregated by request level.

Aggregated Metrics Group	Description
By Risk Classification	Requests that Amazon Cognito identifies as risky.
By Request Classification	Metrics aggregated by request.

Dimensions for Amazon Cognito user pools

The following dimensions are used to refine the usage metrics that are published by Amazon Cognito. The dimensions only apply to `CallCount` and `ThrottleCount` metrics.

Dimension	Description
Service	The name of the AWS service containing the resource. For Amazon Cognito usage metrics, the value for this dimension is `Cognito user pool`.
Type	The type of entity that is being reported. The only valid value for Amazon Cognito usage metrics is API.
Resource	The type of resource that is running. The only valid value is category name.
Class	The class of resource being tracked. Amazon Cognito doesn't use the class dimension.

Use the CloudWatch console to track metrics

You can track and collect Amazon Cognito user pools metrics using CloudWatch. The CloudWatch dashboard will display metrics about every AWS service you use. You can use CloudWatch to create

metric alarms. The alarms can be set up to send you notifications or make a change to a specific resource that you are monitoring. To view service quota metrics in CloudWatch, complete the following steps.

1. Open the CloudWatch console.
2. In the navigation pane, choose **Metrics**.
3. In **All metrics** select a metric and a dimension.
4. Select the check box next to a metric. The metrics will appear in the graph.

> **Note**
>
> Metrics that haven't had any new data points in the past two weeks don't appear in the console. They also don't appear when you enter their metric name or dimension names in the search box in the All metrics tab in the console, and they are not returned in the results of a list-metrics command. The best way to retrieve these metrics is with the `get-metric-data` or `get-metric-statistics` commands in the AWS CLI.

Create a CloudWatch alarm for a quota

Amazon Cognito provides CloudWatch usage metrics that correspond to the AWS service quotas for `CallCount` and `ThrottleCount` APIs. For more information about tracking usage in CloudWatch, see Track quota usage.

In the Service Quotas console, you can create alarms that alert you when your usage approaches a service quota. To learn how to set up a CloudWatch alarm using the Service Quotas console, see Service Quotas and CloudWatch alarms.

Metrics in Service Quotas

You can view and manage your Amazon Cognito user pools and identity pools quotas from a central location with Service Quotas. You can use the Service Quotas console to view details about a specific quota, monitor quota utilization, and request a quota increase. For some quota types, you can create a CloudWatch alarm to track your quota utilization. To learn more about what Amazon Cognito metrics you can track, see Track quota usage.

To view Amazon Cognito user pools and identity pools service quotas utilization, complete the following steps.

1. Open the Service Quotas console.
2. In the navigation pane, choose **AWS services**.
3. From the **AWS services** list, search and choose **Amazon Cognito user pools** or **Amazon Cognito Federated Identities**. The service quota page appears.
4. Select a quota that supports CloudWatch monitoring. For example, choose `Rate of UserAuthentication requests` in Amazon Cognito user pools.
5. Scroll down to **Monitoring**. This section appears only for quotas that support CloudWatch monitoring.
6. In **Monitoring** you can view current service quota utilization in the graph.
7. In **Monitoring** select either one hour, three hours, twelve hours, one day, three days, or one week.
8. Select any area inside of the graph to view the service quota utilization percentage. From here, you can add the graph to your dashboard or use the action menu to select **View in metrics**, which will take you to the related metrics in the CloudWatch console.

Amazon Cognito logging in AWS CloudTrail

Amazon Cognito is integrated with AWS CloudTrail, a service that provides a record of actions taken by a user, role, or an AWS service in Amazon Cognito. CloudTrail captures a subset of API calls for Amazon Cognito as events, including calls from the Amazon Cognito console and from code calls to the Amazon Cognito API operations. If you create a trail, you can choose to deliver CloudTrail events to an Amazon S3 bucket, including events for Amazon Cognito. If you don't configure a trail, you can still view the most recent events in the CloudTrail console in **Event history**. Using the information collected by CloudTrail, you can determine the request that was made to Amazon Cognito, the IP address from which the request was made, who made the request, when it was made, and additional details.

To learn more about CloudTrail, including how to configure and activate it, see the AWS CloudTrail User Guide.

You can also create Amazon CloudWatch alarms for specific CloudTrail events. For example, you can set up CloudWatch to trigger an alarm if an identity pool configuration is changed. For more information, see Creating CloudWatch alarms for CloudTrail events: Examples.

Topics

- Information that Amazon Cognito sends to CloudTrail

- Analyzing Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights
- Example Amazon Cognito events

Information that Amazon Cognito sends to CloudTrail

CloudTrail is turned on when you create your AWS account. When supported event activity occurs in Amazon Cognito, that activity is recorded in a CloudTrail event along with other AWS service events in **Event history**. You can view, search, and download recent events in your AWS account. For more information, see Viewing events with CloudTrail event history.

For an ongoing record of events in your AWS account, including events for Amazon Cognito, create a trail. A CloudTrail trail delivers log files to an Amazon S3 bucket. By default, when you create a trail in the console, the trail applies to all Regions. The trail logs events from all Regions in the AWS partition and delivers the log files to the Amazon S3 bucket that you specify. Additionally, you can configure other AWS services to further analyze and act upon the event data collected in CloudTrail logs. For more information, see:

- Overview for creating a trail
- CloudTrail supported services and integrations
- Configuring amazon SNS notifications for CloudTrail
- Receiving CloudTrail log files from multiple regions and Receiving CloudTrail log files from multiple accounts

Every event or log entry contains information about who generated the request. The identity information helps you determine the following:

- Whether the request was made with root or IAM user credentials.
- Whether the request was made with temporary security credentials for a role or federated user.
- Whether the request was made by another AWS service.

For more information, see the CloudTrail userIdentity element.

Confidential data in AWS CloudTrail

Because user pools and identity pools process user data, Amazon Cognito obscures some private fields in your CloudTrail events with the value `HIDDEN_DUE_TO_SECURITY_REASONS`. For examples of fields that Amazon Cognito doesn't populate to events, see Example Amazon Cognito

events. Amazon Cognito only obscures some fields that commonly contain user information, like passwords and tokens. Amazon Cognito doesn't perform any automatic detection or masking of personally-identifying information that you populate to non-private fields in your API requests.

User pool events

Amazon Cognito supports logging for all of the actions listed on the User pool actions page as events in CloudTrail log files. Amazon Cognito logs user pool events to CloudTrail as *management events*.

The `eventType` field in a Amazon Cognito user pools CloudTrail entry tells you whether your app made the request to the Amazon Cognito user pools API or to an endpoint that serves resources for OpenID Connect, SAML 2.0, or managed login pages. API requests have an `eventType` of `AwsApiCall` and endpoint requests have an `eventType` of `AwsServiceEvent`.

Amazon Cognito logs the following requests to your managed login services as events in CloudTrail.

Hosted UI (classic) events

Hosted UI (classic) events in CloudTrail

Operation	Description
`Login_GET` , `CognitoAuthentication`	A user views or submits credentials to your Login endpoint.
`OAuth2_Authorize_GET` , `Beta_Authorize_GET`	A user views your Authorize endpoint.
`OAuth2Response_GET` , `OAuth2Response_POST`	A user submits an IdP token to your `/oauth2/idpresponse` endpoint.
`SAML2Response_POST` , `Beta_SAML2Response_POST`	A user submits an IdP SAML assertion to your `/saml2/idpresponse` endpoint.
`Login_OIDC_SAML_POST`	A user enters a username at your Login endpoint and matches with an IdP identifier.
`Token_POST` , `Beta_Token_POST`	A user submits an authorization code to your Token endpoint.

Operation	Description
`Signup_GET` , `Signup_POST`	A user submits sign-up information to your `/signup` endpoint.
`Confirm_GET` , `Confirm_POST`	A user submits a confirmation code in the hosted UI.
`ResendCode_POST`	A user submits a request to resend a confirmation code in the hosted UI.
`ForgotPassword_GET` , `ForgotPassword_POST`	A user submits a request to reset their password to your `/forgotPassword` endpoint.
`ConfirmForgotPassword_GET` , `ConfirmForgotPassword_POST`	A user submits a code to your `/confirmForgotPassword` endpoint that confirms their `ForgotPassword` request.
`ResetPassword_GET` , `ResetPassword_POST`	A user submits a new password in the hosted UI.
`Mfa_GET`, `Mfa_POST`	A user submits a multi-factor authentication (MFA) code in the hosted UI.
`MfaOption_GET` , `MfaOption_POST`	A user chooses their preferred method for MFA in the hosted UI.
`MfaRegister_GET` , `MfaRegister_POST`	A user submits a multi-factor authentication (MFA) code in the hosted UI when registering the MFA.
`Logout`	A user signs out at your `/logout` endpoint.
`SAML2Logout_POST`	A user signs out at your `/saml2/logout` endpoint.
`Error_GET`	A user views an error page in the hosted UI.

Operation	Description
`UserInfo_GET` , `UserInfo_POST`	A user or IdP exchanges information with your userInfo endpoint.
`Confirm_With_Link_GET`	A user submits a confirmation based on a link that Amazon Cognito sent in an email message.
`Event_Feedback_GET`	A user submits feedback to Amazon Cognito about a threat protection event.

Managed login events

Managed login events in CloudTrail

Operation	Description
`login_POST`	A user submits credentials to your Login endpoint.
`login_continue_POST`	A user who has already signed in one time chooses to sign in again.
`forgotPassword_POST`	A user resets their password.
`selectChallenge_POST`	A user responds to an authentication challenge after they submit their username or credentials.
`confirmUser_GET`	A user opens the link in a confirmation or verification email message.
`mfa_back_POST`	A user chooses the **Back** button after an MFA prompt.
`mfa_options_POST`	A user selects an MFA option.
`mfa_phone_register_POST`	A user submits a phone number to register as a MFA factor. This operation causes

Operation	Description
	Amazon Cognito to send an MFA code to their phone number.
`mfa_phone_verify_POST`	A user submits an MFA code sent to their phone number.
`mfa_phone_resendCode_POST`	A user submits a request to resend a MFA code to their phone number.
`mfa_totp_POST`	A user submits a TOTP MFA code.
`signup_POST`	A user submits information to your `/signup` managed login page.
`signup_confirm_POST`	A user submits a confirmation code from an email or SMS message.
`verifyCode_POST`	A user submits a one-time password (OTP) for passwordless authentication.
`passkeys_add_POST`	A user submits a request to register a new passkey credential.
`passkeys_add_GET`	A user navigates to the page where they can register a passkey.
`login_passkey_POST`	A user signs in with a passkey.

Note

Amazon Cognito records `UserSub` but not `UserName` in CloudTrail logs for requests that are specific to a user. You can find a user for a given `UserSub` by calling the `ListUsers` API, and using a filter for sub.

Identity pools events

Data events

Amazon Cognito logs the following Amazon Cognito Identity events to CloudTrail as *data events*. Data events are high-volume data-plane API operations that CloudTrail doesn't log by default. Additional charges apply for data events.

- GetCredentialsForIdentity
- GetId
- GetOpenIdToken
- GetOpenIdTokenForDeveloperIdentity
- UnlinkIdentity

To generate CloudTrail logs for these API operations, you must activate data events in your trail and choose event selectors for **Cognito identity pools**. For more information, see Logging data events for trails in the *AWS CloudTrail User Guide*.

You can also add identity pools event selectors to your trail with the following CLI command.

```
aws cloudtrail put-event-selectors --trail-name <trail name> --advanced-event-selectors
 \
"{\
   \"Name\": \"Cognito Selector\",\
   \"FieldSelectors\": [\
      {\
         \"Field\": \"eventCategory\",\
         \"Equals\": [\
            \"Data\"\
         ]\
      },\
      {\
         \"Field\": \"resources.type\",\
         \"Equals\": [\
            \"AWS::Cognito::IdentityPool\"\
         ]\
      }\
   ]\
}"
```

Management events

Amazon Cognito logs the remainder of Amazon Cognito identity pools API operations as *management events*. CloudTrail logs management event API operations by default.

For a list of the Amazon Cognito identity pools API operations that Amazon Cognito logs to CloudTrail, see the Amazon Cognito identity pools API Reference.

Amazon Cognito Sync

Amazon Cognito logs all Amazon Cognito Sync API operations as management events. For a list of the Amazon Cognito Sync API operations that Amazon Cognito logs to CloudTrail, see the Amazon Cognito Sync API Reference.

Analyzing Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights

You can search and analyze your Amazon Cognito CloudTrail events with Amazon CloudWatch Logs Insights. When you configure your trail to send events to CloudWatch Logs, CloudTrail sends only the events that match your trail settings.

To query or research your Amazon Cognito CloudTrail events, in the CloudTrail console, make sure that you select the **Management events** option in your trail settings so that you can monitor the management operations performed on your AWS resources. You can optionally select the **Insights events** option in your trail settings when you want to identify errors, unusual activity, or unusual user behavior in your account.

Sample Amazon Cognito queries

You can use the following queries in the Amazon CloudWatch console.

General queries

Find the 25 most recently added log events.

```
fields @timestamp, @message | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com"
```

Get a list of the 25 most recently added log events that include exceptions.

```
fields @timestamp, @message | sort @timestamp desc | limit 25
```

```
| filter eventSource = "cognito-idp.amazonaws.com" and @message like /Exception/
```

Exception and Error Queries

Find the 25 most recently added log events with error code `NotAuthorizedException` along with Amazon Cognito user pool `sub`.

```
fields @timestamp, additionalEventData.sub as user | sort @timestamp desc | limit 25
| filter eventSource = "cognito-idp.amazonaws.com" and errorCode=
 "NotAuthorizedException"
```

Find the number of records with `sourceIPAddress` and corresponding `eventName`.

```
filter eventSource = "cognito-idp.amazonaws.com"
| stats count(*) by sourceIPAddress, eventName
```

Find the top 25 IP addresses that triggered a `NotAuthorizedException` error.

```
filter eventSource = "cognito-idp.amazonaws.com" and errorCode=
 "NotAuthorizedException"
| stats count(*) as count by sourceIPAddress, eventName
| sort count desc | limit 25
```

Find the top 25 IP addresses that called the `ForgotPassword` API.

```
filter eventSource = "cognito-idp.amazonaws.com" and eventName = 'ForgotPassword'
| stats count(*) as count by sourceIPAddress
| sort count desc | limit 25
```

Example Amazon Cognito events

Amazon Cognito logs information to AWS CloudTrail about user authentication activity and administrative management activity. This applies to both user pools and identity pools.
For example, you can see `GetId` and `UpdateIdentityPool` events in the same trail, or `UpdateAuthEventFeedback` and `SetRiskConfiguration` events. You'll also see user pool logs for hosted UI activity that doesn't correspond to operations in the user pools API. This section has some examples of logs you might see. To understand the CloudTrail event schema for any operation, generate a request for that operation and review the events that it creates in your trail.

A trail can deliver events as log files to an Amazon S3 bucket that you specify. CloudTrail log files contain one or more log entries. An event represents a single request from any source and includes information about the requested action, the date and time of the action, request parameters, and so on. CloudTrail log files are not an ordered stack trace of the public API calls, so they do not appear in any specific order.

Topics

- Example CloudTrail events for a hosted UI sign-up
- Example CloudTrail event for a SAML request
- Example CloudTrail events for requests to the token endpoint
- Example CloudTrail event for CreateIdentityPool
- Example CloudTrail event for GetCredentialsForIdentity
- Example CloudTrail event for GetId
- Example CloudTrail event for GetOpenIdToken
- Example CloudTrail event for GetOpenIdTokenForDeveloperIdentity
- Example CloudTrail event for UnlinkIdentity

Example CloudTrail events for a hosted UI sign-up

The following example CloudTrail events demonstrate the information that Amazon Cognito logs when a user signs up through the hosted UI.

Amazon Cognito logs the following event when a new user navigates to the sign-in page for your app.


```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-04-06T05:38:12Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Login_GET",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "errorCode": "",
```

```
    "errorMessage": "",
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200.0
        },
        "requestParameters":
        {
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "response_type":
            [
                "token"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        }
    },
    "eventID": "382ae09a-151d-4116-8f2b-6ac0a804a38c",
    "readOnly": true,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```


Amazon Cognito logs the following event when a new user chooses **Sign up** from the sign-in page for your app.


```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
```

```
    },
    "eventTime": "2022-05-05T23:21:43Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Signup_GET",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200
        },
        "requestParameters":
        {
            "response_type":
            [
                "code"
            ],
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "7a63e7c2-b057-4f3d-a171-9d9113264fff",
    "eventID": "5e7b27a0-6870-4226-adb4-f86cd51ac5d8",
    "readOnly": true,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
```

```
}
```

Amazon Cognito logs the following event when a new user chooses a username, enters an email address, and chooses a password from the sign-in page for your app. Amazon Cognito doesn't log identifying information about the user's identity to CloudTrail.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-05T23:22:05Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Signup_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 302
        },
        "requestParameters":
        {
            "password":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "requiredAttributes[email]":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "response_type":
            [
                "code"
            ],
            "_csrf":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
```

```
            ],
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "client_id":
            [
                "1example23456789"
            ],
            "username":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "9ad58dd8-3517-4aa8-96a5-d17a01df9eb4",
    "eventID": "c75eb7a5-eb8c-43d1-8331-f4412e756e69",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```

Amazon Cognito logs the following event when a new user accesses the user confirmation page in the hosted UI after they sign up.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-05T23:22:06Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Confirm_GET",
    "awsRegion": "us-west-2",
```

```
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200
        },
        "requestParameters":
        {
            "response_type":
            [
                "code"
            ],
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "58a5b170-3127-45bb-88cc-3e652d779e0b",
    "eventID": "7f87291a-6d50-409a-822f-e3a5ec7e60da",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```


Amazon Cognito logs the following event when, in the user confirmation page in the hosted UI, a user enters a code that Amazon Cognito sent them in an email message.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-05T23:23:32Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Confirm_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 302
        },
        "requestParameters":
        {
            "confirm":
            [
                ""
            ],
            "deliveryMedium":
            [
                "EMAIL"
            ],
            "sub":
            [
                "704b1e47-34fe-40e9-8c41-504997494531"
            ],
            "code":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "destination":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "response_type":
```

```
            [
                "code"
            ],
            "_csrf":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "cognitoAsfData":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "client_id":
            [
                "1example23456789"
            ],
            "username":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "9764300a-ed35-4f87-8a0f-b18b3fe2b11e",
    "eventID": "e24ac6e5-2f70-4c6e-ad4e-2f08a547bb36",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```

Example CloudTrail event for a SAML request

Amazon Cognito logs the following event when a user who has authenticated with your SAML IdP submits the SAML assertion to your `/saml2/idpresponse` endpoint.


```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-06T00:50:57Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "SAML2Response_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 302
        },
        "requestParameters":
        {
            "RelayState":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "SAMLResponse":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "4f6f15d1-c370-4a57-87f0-aac4817803f7",
    "eventID": "9824b50f-d9d1-4fb8-a2c1-6aa78ca5902a",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
```

```
    "recipientAccountId": "625647942648",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```

Example CloudTrail events for requests to the token endpoint

The following are example events from requests to the Token endpoint.

Amazon Cognito logs the following event when a user who has authenticated and received an authorization code submits the code to your `/oauth2/token` endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-12T22:12:30Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Token_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200
        },
        "requestParameters":
        {
            "code":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "grant_type":
            [
```

```
                "authorization_code"
            ],
            "redirect_uri":
            [
                "https://www.amazon.com"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "f257f752-cc14-4c52-ad5b-152a46915238",
    "eventID": "0bd1586d-cd3e-4d7a-abaf-fd8bfc3912fd",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```

Amazon Cognito logs the following event when your backend system submits a `client_credentials` request for an access token to your `/oauth2/token` endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
    "eventTime": "2022-05-12T21:07:05Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Token_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
```

```
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200
        },
        "requestParameters":
        {
            "grant_type":
            [
                "client_credentials"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "4f871256-6825-488a-871b-c2d9f55caff2",
    "eventID": "473e5cbc-a5b3-4578-9ad6-3dfdcb8a6d34",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
}
```

Amazon Cognito logs the following event when your app exchanges a refresh token for a new ID and access token with your `/oauth2/token` endpoint.

```
{
    "eventVersion": "1.08",
    "userIdentity":
    {
        "accountId": "123456789012"
    },
```

```
    "eventTime": "2022-05-12T22:16:40Z",
    "eventSource": "cognito-idp.amazonaws.com",
    "eventName": "Token_POST",
    "awsRegion": "us-west-2",
    "sourceIPAddress": "192.0.2.1",
    "userAgent": "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_15_7)...",
    "requestParameters": null,
    "responseElements": null,
    "additionalEventData":
    {
        "responseParameters":
        {
            "status": 200
        },
        "requestParameters":
        {
            "refresh_token":
            [
                "HIDDEN_DUE_TO_SECURITY_REASONS"
            ],
            "grant_type":
            [
                "refresh_token"
            ],
            "client_id":
            [
                "1example23456789"
            ]
        },
        "userPoolDomain": "mydomain.auth.us-west-2.amazoncognito.com",
        "userPoolId": "us-west-2_EXAMPLE"
    },
    "requestID": "2829f0c6-a3a9-4584-b046-11756dfe8a81",
    "eventID": "12bd3464-59c7-44fa-b8ff-67e1cf092018",
    "readOnly": false,
    "eventType": "AwsServiceEvent",
    "managementEvent": true,
    "recipientAccountId": "123456789012",
    "serviceEventDetails":
    {
        "serviceAccountId": "111122223333"
    },
    "eventCategory": "Management"
```

```
}
```

Example CloudTrail event for CreateIdentityPool

The following example is a log entry for a request for the `CreateIdentityPool` action. The request was made by an IAM user named Alice.

```
{
    "eventVersion": "1.03",
    "userIdentity": {
        "type": "IAMUser",
        "principalId": "PRINCIPAL_ID",
        "arn": "arn:aws:iam::123456789012:user/Alice",
        "accountId": "123456789012",
        "accessKeyId": "['EXAMPLE_KEY_ID']",
        "userName": "Alice"
    },
    "eventTime": "2016-01-07T02:04:30Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "CreateIdentityPool",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "127.0.0.1",
    "userAgent": "USER_AGENT",
    "requestParameters": {
        "identityPoolName": "TestPool",
        "allowUnauthenticatedIdentities": true,
        "supportedLoginProviders": {
            "graph.facebook.com": "000000000000000"
        }
    },
    "responseElements": {
        "identityPoolName": "TestPool",
        "identityPoolId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
        "allowUnauthenticatedIdentities": true,
        "supportedLoginProviders": {
            "graph.facebook.com": "000000000000000"
        }
    },
    "requestID": "15cc73a1-0780-460c-91e8-e12ef034e116",
    "eventID": "f1d47f93-c708-495b-bff1-cb935a6064b2",
    "eventType": "AwsApiCall",
    "recipientAccountId": "123456789012"
}
```

Example CloudTrail event for GetCredentialsForIdentity

The following example is a log entry for a request for the `GetCredentialsForIdentity` action.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "Unknown"
    },
    "eventTime": "2023-01-19T16:55:08Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "GetCredentialsForIdentity",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.4",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off
 command/cognito-identity.get-credentials-for-identity",
    "requestParameters": {
        "logins": {
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaa":
 "HIDDEN_DUE_TO_SECURITY_REASONS"
        },
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
    },
    "responseElements": {
        "credentials": {
            "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
            "sessionToken": "aAaAaAaAaAaAab1111111111EXAMPLE",
            "expiration": "Jan 19, 2023 5:55:08 PM"
        },
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
    },
    "requestID": "659dfc23-7c4e-4e7c-858a-1abce884d645",
    "eventID": "6ad1c766-5a41-4b28-b5ca-e223ccb00f0d",
    "readOnly": false,
    "resources": [{
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
```

```
}
```

Example CloudTrail event for GetId

The following example is a log entry for a request for the `GetId` action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "Unknown"
    },
    "eventTime": "2023-01-19T16:55:05Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "GetId",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.4",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off
 command/cognito-identity.get-id",
    "requestParameters": {
        "identityPoolId": "us-east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE",
        "logins": {
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaa":
 "HIDDEN_DUE_TO_SECURITY_REASONS"
        }
    },
    "responseElements": {
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
    },
    "requestID": "dc28def9-07c8-460a-a8f3-3816229e6664",
    "eventID": "c5c459d9-40ec-41fd-8f6b-57865d5a9975",
    "readOnly": false,
    "resources": [{
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
}
```

Example CloudTrail event for GetOpenIdToken

The following example is a log entry for a request for the `GetOpenIdToken` action.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "Unknown"
    },
    "eventTime": "2023-01-19T16:55:08Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "GetOpenIdToken",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.4",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off
 command/cognito-identity.get-open-id-token",
    "requestParameters": {
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
        "logins": {
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaa":
 "HIDDEN_DUE_TO_SECURITY_REASONS"
        }
    },
    "responseElements": {
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
    },
    "requestID": "a506ba18-10d7-4fdb-9548-a8187b2e38bb",
    "eventID": "19ffc1a6-6ed8-4580-a4e1-3062c5ce6457",
    "readOnly": false,
    "resources": [{
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
}
```

Example CloudTrail event for GetOpenIdTokenForDeveloperIdentity

The following example is a log entry for a request for the `GetOpenIdTokenForDeveloperIdentity` action.


```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "AssumedRole",
        "principalId": "AROA1EXAMPLE:johns-AssumedRoleSession",
        "arn": "arn:aws:sts::111122223333:assumed-role/Admin/johns-AssumedRoleSession",
        "accountId": "111122223333",
        "accessKeyId": "ASIAIOSFODNN7EXAMPLE",
        "sessionContext": {
            "sessionIssuer": {
                "type": "Role",
                "principalId": "AROA1EXAMPLE",
                "arn": "arn:aws:iam::111122223333:role/Admin",
                "accountId": "111122223333",
                "userName": "Admin"
            },
            "attributes": {
                "creationDate": "2023-01-19T16:53:14Z",
                "mfaAuthenticated": "false"
            }
        }
    },
    "eventTime": "2023-01-19T16:55:08Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "GetOpenIdTokenForDeveloperIdentity",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "27.0.3.154",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off
 command/cognito-identity.get-open-id-token-for-developer-identity",
    "requestParameters": {
        "tokenDuration": 900,
        "identityPoolId": "us-east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE",
        "logins": {
            "JohnsDeveloperProvider": "HIDDEN_DUE_TO_SECURITY_REASONS"
        }
    },
    "responseElements": {
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE"
    },
```

```
    "requestID": "b807df87-57e7-4dd6-b90c-b06f46a61c21",
    "eventID": "f26fed91-3340-4d70-91ae-cdf555547b76",
    "readOnly": false,
    "resources": [{
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
}
```

Example CloudTrail event for UnlinkIdentity

The following example is a log entry for a request for the `UnlinkIdentity` action.

```
{
    "eventVersion": "1.08",
    "userIdentity": {
        "type": "Unknown"
    },
    "eventTime": "2023-01-19T16:55:08Z",
    "eventSource": "cognito-identity.amazonaws.com",
    "eventName": "UnlinkIdentity",
    "awsRegion": "us-east-1",
    "sourceIPAddress": "192.0.2.4",
    "userAgent": "aws-cli/2.7.25 Python/3.9.11 Darwin/21.6.0 exe/x86_64 prompt/off
 command/cognito-identity.unlink-identity",
    "requestParameters": {
        "logins": {
            "cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaa":
 "HIDDEN_DUE_TO_SECURITY_REASONS"
        },
        "identityId": "us-east-1:1cf667a2-49a6-454b-9e45-23199EXAMPLE",
        "loginsToRemove": ["cognito-idp.us-east-1.amazonaws.com/us-east-1_aaaaaaaaa"]
    },
    "responseElements": null,
    "requestID": "99c2c8e2-9c29-416f-bb17-b650a5cbada9",
    "eventID": "d8e26126-202a-43c2-b458-3f225efaedc7",
    "readOnly": false,
    "resources": [{
```

```
        "accountId": "111122223333",
        "type": "AWS::Cognito::IdentityPool",
        "ARN": "arn:aws:cognito-identity:us-east-1:111122223333:identitypool/us-
east-1:2dg778b3-50b7-565c-0f56-34200EXAMPLE"
    }],
    "eventType": "AwsApiCall",
    "managementEvent": false,
    "recipientAccountId": "111122223333",
    "eventCategory": "Data"
}
```

Access Amazon Cognito using an interface endpoint (AWS PrivateLink)

You can use AWS PrivateLink to create a private connection between your VPC and Amazon Cognito. You can access Amazon Cognito as if it were in your VPC, without the use of an internet gateway, NAT device, VPN connection, or Direct Connect connection. Instances in your VPC don't need public IP addresses to access Amazon Cognito.

You establish this private connection by creating an *interface endpoint*, powered by AWS PrivateLink. We create an endpoint network interface in each subnet that you enable for the interface endpoint. These are requester-managed network interfaces that serve as the entry point for traffic destined for Amazon Cognito.

For more information, see Access AWS services through AWS PrivateLink in the *AWS PrivateLink Guide*.

Important

The following authentication types aren't currently supported through AWS PrivateLink:

1. Machine to machine (M2M) authorization with the OAuth 2.0 client credentials flow from the token endpoint. M2M authorization with the `GetClientToken` API operation is supported, because it doesn't require a user pool domain.
2. Sign-in with managed login and the classic hosted UI.

Topics

- Authentication flows for AWS PrivateLink integration

- Operational modes for AWS PrivateLink
- Considerations for Amazon Cognito
- Controlling access with resource control policies
- Create an interface endpoint for Amazon Cognito
- Create an endpoint policy for your interface endpoint
- Create an identity-based policy for AWS PrivateLink operations

Authentication flows for AWS PrivateLink integration

The following table describes the authentication flows available to clients in VPCs, and the IAM policies that you can apply to govern them. The policies that you can evaluate in requests to user pools are resource control policies (RCPs), VPC endpoint policies, and identity-based policies.

Resource	Authentication flow	Policies evaluated when client transits a VPC endpoint	Policies evaluated when client origin is public
User pool	Managed login & classic hosted UI sign-in	None (no access)[1]	None[2]
User pool	Machine-to-machine authorization from the token endpoint (M2M authorization with the `GetClientToken` API operation is covered by the SDK and REST API unauthenticated requests row)	None (no access)[1]	None[2]
User pool	SDK and REST API unauthenticated requests	RCPs, VPC endpoint policies[3]	RCPs
User pool	SDK and REST API SigV4 authenticated requests	RCPs, VPC endpoint policies, identity-based policies[3]	RCPs, identity-based policies

Resource	Authentication flow	Policies evaluated when client transits a VPC endpoint	Policies evaluated when client origin is public
Identity pool	SDK and REST API unauthenticated requests (basic and enhanced flows)	RCPs, VPC endpoint policies	RCPs
Identity pool	SDK and REST API SigV4 authenticated requests (developer-authenticated flow)	RCPs, identity-based policies	RCPs, identity-based policies

[1] VPC endpoints don't accept requests for user pool domains. If the client has a route to the internet, NAT is applied, making the origin public.

[2] The existence of a user pool domain prevents completion of any user pool requests that transit a VPC endpoint. Any client can take public transit paths *only* to the user pool domain and API service endpoints, making the VPC endpoint unusable for the user pool. User pools with domains assigned are incompatible with AWS PrivateLink.

[3] User pool must *not* have a domain assigned.

Operational modes for AWS PrivateLink

The following example implementation models are supported with AWS PrivateLink and Amazon Cognito.

Resource	Implementation	Actions
User pool	Fully private SDK or REST API application	1. Delete domain 2. Create VPC endpoint 3. Configure RCP to Deny all cognito-idp actions except from VPC
User pool	Private and public	1. Delete domain 2. Create VPC endpoint

Resource	Implementation	Actions
User pool	Private or public OAuth 2.0 authorization server	1. Not available to VPC
Identity pool	Fully private	1. Create VPC endpoint 2. Configure RCP to Deny all cognito-identity actions except from VPC
Identity pool	Private and public	1. Create VPC endpoint

Considerations for Amazon Cognito

Before you set up an interface endpoint for Amazon Cognito, review Considerations in the *AWS PrivateLink Guide*. Amazon Cognito supports making calls to all Amazon Cognito API actions through the interface endpoint. For more information about these operations, see the Amazon Cognito user pools API Reference and Amazon Cognito Federated Identities API Reference.

Topics

- User pools and AWS PrivateLink
- Identity pools and AWS PrivateLink

User pools and AWS PrivateLink

You can make requests to all user pools API operations through the interface endpoint, but not to operations that your application requests from the user pool OAuth 2.0 authorization server for example client credentials grants and managed login.

The `cognito-idp` user pools API has unauthenticated, authenticated, and token-authorized API operations. You can grant permissions for authenticated operations in VPC endpoint and resource control policies. You can *also* grant permissions for unauthenticated and token-authorized operations, unlike in identity-based policies. VPC endpoint and resource control policy types are able to evaluate and deny or allow requests for otherwise-public operations.

Requests to domain endpoints are also public, but you can't evaluate them in policies. VPC private DNS doesn't route requests for user pool domains to your VPC endpoint. You can only make

requests for domain services through public-internet paths. For more information, see Effects of policies on user pool operations.

Topics

- Supported operations
- Effects of policies on user pool operations

Supported operations

Systems in a VPC can send requests to user pool API actions but *not* to user pool domain endpoints. OpenID Connect (OIDC) and OAuth 2.0 workflows that use domain endpoints, for example machine-to-machine (M2M), federated sign-in, and authorization code grants, are inaccessible through VPC endpoints. VPC endpoint policies have no effect on these HTTP workflows and can't process them. Requests to domain endpoints from within a VPC always fail at the interface endpoint, but continue to be available through public DNS and routing when you set up VPC endpoints for your user pools.

To prevent the assignment of domains from systems in a VPC, Amazon Cognito blocks `CreateUserPoolDomain` requests at the interface endpoint. This prevents addition of domains to your user pools from systems that are in a VPC. To prevent the addition of a domain from all systems, apply a resource control policy (RCP) like the following example to your AWS account. This policy blocks the `CreateUserPoolDomain` action against the specified user pool.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Principal": "*",
            "Effect": "Deny",
            "Action": [
                "cognito-idp:CreateUserPoolDomain"
            ],
            "Resource": "arn:aws:cognito-idp:eusc-de-east-1:123456789012:userpool/eusc-de-east-1_EXAMPLE"
        }
    ]
}
```

Your user pool might have a domain, and in all cases that domain is unavailable through AWS PrivateLink. All SDK-based user pool API requests to `cognito-idp` service endpoints accept requests through AWS PrivateLink, with the exception of `CreateUserPoolDomain`. User pool API service endpoints and domain endpoints remain always accessible through public-internet paths. To address access from public sources, implement AWS WAF web ACLs.

Effects of policies on user pool operations

All user pool API operations, even those that are typically public and unauthenticated, can be controlled in VPC endpoint policies and resource control policies (RCPs). You can also apply restrictions to user pool access in identity-based policies with VPC condition keys. Only requests that include authentication information in SigV4 format can be controlled in identity-based policies. Managed login and classic hosted UI operations are a separate category, and aren't eligible for VPC transit or the application of any type of policies to their actions.

Unauthenticated operations

Amazon Cognito operations for client-side applications aren't authenticated with SigV4. Example operations are in the example policy at Create an endpoint policy for your interface endpoint. Additional examples of unauthenticated operations are `GetUser` and `AssociateSoftwareToken`. When you add these operations to identity-based policies, they have no effect. However, you can permit or restrict access to these actions in VPC endpoint policies and RCPs.

Unauthenticated operations aren't associated with an IAM principal. Your VPC endpoint policy or RCP must allow all principals for these actions.

Authenticated operations

API operations for user pool administration and server-side authentication *are* authenticated with SigV4. For authenticated operations, you can restrict principals with endpoint policies that you apply to the VPC endpoint, resource control policies in your organization, and in identity-based policies that you apply to principals. Identity-based and resource-control policies are VPC-aware with network-based condition keys like `aws:SourceVpc` and `aws:SourceVpce`.

For more information about server-side, client-side, and administrative classes of API operations for user pools, see Authorization models for API and SDK authentication.

Identity pools and AWS PrivateLink

Amazon Cognito identity pools support all API operations through AWS PrivateLink.

Topics

- Supported operations
- Network context limitations with AWS STS integration
- Service-specific context keys

Supported operations

All identity pools API operations are supported through the interface endpoint. Identity pools don't have domain endpoints and aren't subject to the same limitations. However, identity pools have specific considerations for network-based access controls due to their integration with AWS STS.

Network context limitations with AWS STS integration

Identity pools use AWS STS `AssumeRoleWithWebIdentity` operations to provide temporary AWS credentials. When identity pools call AWS STS through AWS PrivateLink in the enhanced authentication flow, network context keys like `aws:SourceIp`, `aws:SourceVpc`, and `aws:SourceVpce` contain values from the identity pools service infrastructure, not from your application's network context.

If your IAM role trust policies or resource control policies (RCPs) use network-based condition keys to restrict access, identity pools operations might be denied unexpectedly. To address this limitation, you can use one of the following approaches:

Principal tags for service identification

Tag the IAM roles used with identity pools and modify your policies to allow operations when the principal has the appropriate tag. First, add a tag to your identity pool role:

```
aws iam tag-role \
    --role-name MyIdentityPoolRole \
    --tags Key=CognitoServiceCall,Value=true
```

Then modify your network-based policies to allow tagged principals. For example, in an RCP:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Deny",
            "Principal": "*",
```

```
            "Action": "sts:AssumeRoleWithWebIdentity",
            "Resource": "*",
            "Condition": {
                "NotIpAddress": {
                    "aws:SourceIp": ["allowed-ip-ranges"]
                },
                "StringNotEqualsIfExists": {
                    "aws:ResourceTag/CognitoServiceCall": "true"
                }
            }
        }
    ]
}
```

Service-specific context keys

Identity pools provide service-specific context keys for resource-level authorization in VPC endpoint policies and RCPs. With these context keys, you can enable fine-grained access control and distinguish between authenticated and unauthenticated users in policies.

Available service-specific context keys for non-SigV4 operations like GetId, GetCredentialsForIdentity, GetOpenIdToken, UnlinkIdentity

- `cognito-identity-unauth:IdentityPoolArn` - Filters access by the identity pool ARN for unauthenticated users
- `cognito-identity-unauth:AccountId` - Filters access by the AWS account ID for unauthenticated users
- `cognito-identity-auth:IdentityPoolArn` - Filters access by the identity pool ARN for authenticated users
- `cognito-identity-auth:AccountId` - Filters access by the AWS account ID for authenticated users

Available service-specific context keys for SigV4 operations like DeleteIdentities and DescribeIdentity

- `cognito-identity:IdentityPoolArn` - Filters access by the identity pool ARN

You can use these context keys in VPC endpoint policies to restrict access based on authentication status, as demonstrated in the following example:

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": "*",
            "Action": [
                "cognito-identity:GetId",
                "cognito-identity:GetCredentialsForIdentity"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "cognito-identity-unauth:IdentityPoolArn": "arn:aws:cognito-
identity:eusc-de-east-1:123456789012:identitypool/eusc-de-east-1:12345678-ffff-ffff-
ffff-123456"
                }
            }
        }
    ]
}
```

Controlling access with resource control policies

Amazon Cognito supports controlling access to your resources with resource control policies (RCPs). With network-based condition keys, RCPs can define the networks and actions that are permitted for AWS PrivateLink access to your user pools and identity pools. The `Action` statements in RCPs can control access to both authenticated and unauthenticated user pool API operations.

For example, the following example policy prevents access to all user pools from a specific VPC.

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "DenyCognitoAccessOutsideVPC",
      "Effect": "Deny",
      "Principal": "*",
      "Action": "cognito-idp:*",
      "Resource": "*",
      "Condition": {
        "StringNotEqualsIfExists": {
```

```
          "aws:SourceVpc": "vpc-02d6770f46ef1653b"
        }
      }
    }
  ]
}
```

Create an interface endpoint for Amazon Cognito

You can create an interface endpoint for Amazon Cognito using either the Amazon VPC console or the AWS Command Line Interface (AWS CLI). For more information, see Create an interface endpoint in the *AWS PrivateLink Guide*.

Create an interface endpoint for Amazon Cognito user pools using the following service name:

```
com.amazonaws.region.cognito-idp
```

Create an interface endpoint for Amazon Cognito identity pools using the following service name:

```
com.amazonaws.region.cognito-identity
```

If you enable private DNS for the interface endpoint, you can make API requests to Amazon Cognito using its default Regional DNS name. For example, `cognito-idp.eusc-de-east-1.amazonaws.com` for user pools and `cognito-identity.eusc-de-east-1.amazonaws.com` for identity pools.

Create an endpoint policy for your interface endpoint

An endpoint policy is an IAM resource that you can attach to an interface endpoint. The default endpoint policy allows full access to Amazon Cognito through the interface endpoint. To control the access allowed to Amazon Cognito from your VPC, attach a custom endpoint policy to the interface endpoint.

An endpoint policy specifies the following information:

- The principals that can perform actions (AWS accounts, IAM users, and IAM roles).
- The actions that can be performed.
- The resources on which the actions can be performed.

- The conditions that must be satisfied before the request is allowed or denied.

For more information, see Control access to services using endpoint policies in the *AWS PrivateLink Guide*.

Example: VPC endpoint policy for user pool actions

The following is an example of a custom endpoint policy for user pools. When you attach this policy to your interface endpoint, it grants access to the listed user pool actions for all principals on all resources.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Principal": {
            "AWS": "arn:aws:iam::123456789012:assumed-role/MyWebAppRole/
MyWebAppSession"
         },
         "Effect": "Allow",
         "Action": [
            "cognito-idp:AdminInitiateAuth",
            "cognito-idp:AdminRespondToAuthChallenge",
            "cognito-idp:AdminSetUserPassword"
         ],
         "Resource":"arn:aws:cognito-idp:eusc-de-east-1:123456789012:userpool/eusc-de-
east-1_EXAMPLE"
      },
      {
         "Effect": "Allow",
         "Action": [
            "cognito-idp:InitiateAuth",
            "cognito-idp:RespondToAuthChallenge",
            "cognito-idp:ForgotPassword",
            "cognito-idp:ConfirmForgotPassword"
         ],
         "Resource":"arn:aws:cognito-idp:eusc-de-east-1:123456789012:userpool/eusc-de-
east-1_EXAMPLE"
      }
   ]
}
```

Example: VPC endpoint policy for identity pool actions

The following is an example of a custom endpoint policy for identity pools. This policy uses service-specific context keys to restrict access to authenticated users from a specific identity pool.

```
{
   "Version": "2012-10-17",
   "Statement": [
      {
         "Effect": "Allow",
         "Principal": "*",
         "Action": [
            "cognito-identity:GetId",
            "cognito-identity:GetCredentialsForIdentity",
            "cognito-identity:GetOpenIdToken"
         ],
         "Resource": "*",
         "Condition": {
            "StringEquals": {
               "cognito-identity-auth:IdentityPoolArn": "arn:aws:cognito-identity:eusc-de-east-1:123456789012:identitypool/eusc-de-east-1:12345678-ffff-ffff-ffff-123456"
            }
         }
      }
   ]
}
```

Create an identity-based policy for AWS PrivateLink operations

Identity-based policies are IAM resources that you can attach to AWS principals. You can control access to Amazon Cognito through VPC endpoints with identity-based policies for IAM-authenticated operations. Unlike endpoint policies, you can't configure permissions for unauthenticated operations in identity-based policies. Authenticated, or administrative, operations require Signature Version 4 authorization. For user pools, authenticated operations include server-side authentication requests like AdminInitiateAuth and administrative requests like UpdateUserPool. For identity pools, authenticated operations include administrative requests like DeleteIdentities and DescribeIdentity.

An identity-based policy specifies the following information:

- The actions that can be performed.

- The resources on which the actions can be performed.
- The conditions that must be satisfied before the request is allowed or denied.

Example: identity-based policy for user pool server-side authentication

The following example policy grants access to the listed user pool actions in the specified user pool, from the specified endpoint. Apply this policy to the assumed IAM role for your web application.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Action": [
                "cognito-idp:AdminInitiateAuth",
                "cognito-idp:AdminRespondToAuthChallenge",
                "cognito-idp:AdminSetUserPassword"
            ],
            "Resource": "arn:aws:cognito-idp:eusc-de-east-1:123456789012:userpool/eusc-de-east-1_EXAMPLE",
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d"
                }
            }
        }
    ]
}
```

Example: identity-based policy for identity pool administrative operations

The following example policy grants access to identity pool administrative actions from the specified VPC endpoint. Apply this policy to the IAM principal that needs to perform identity pool administration.

```
{
    "Version": "2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
```

```
            "Action": [
                "cognito-identity:DeleteIdentities",
                "cognito-identity:DescribeIdentity"
            ],
            "Resource": "*",
            "Condition": {
                "StringEquals": {
                    "aws:SourceVpce": "vpce-1a2b3c4d",
                    "cognito-identity:IdentityPoolArn": "arn:aws:cognito-identity:eusc-de-east-1:123456789012:identitypool/eusc-de-east-1:12345678-ffff-ffff-ffff-123456"
                }
            }
        }
    ]
}
```

Compliance validation for Amazon Cognito

Third-party auditors assess the security and compliance of Amazon Cognito as part of multiple AWS compliance programs. These include SOC, PCI, FedRAMP, HIPAA, and others.

For a list of AWS services in scope of specific compliance programs, see . For general information, see .

Your compliance responsibility when using Amazon Cognito is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. AWS provides the following resources to help with compliance:

- – This collection of workbooks and guides might apply to your industry and location.
- Evaluating resources with rules in the *AWS Config Developer Guide* – AWS Config; assesses how well your resource configurations comply with internal practices, industry guidelines, and regulations.

Resilience in Amazon Cognito

The AWS global infrastructure is built around AWS Regions and Availability Zones. Regions provide multiple physically separated and isolated Availability Zones, which are connected through low-latency, high-throughput, and highly redundant networking. With Availability Zones, you can design and operate applications and databases that automatically fail over between zones

without interruption. Availability Zones are more highly available, fault tolerant, and scalable than traditional single or multiple data center infrastructures.

For more information about AWS Regions and Availability Zones, see AWS global infrastructure.

Topics

- Regional data considerations

Regional data considerations

Amazon Cognito user pools are each created in one AWS Region, and they store the user profile data only in that region. User pools can send user data to a different AWS Region, depending on how optional features are configured.

- If the default `no-reply@verificationemail.com` email address setting is used for routing verification of emails addresses with Amazon Cognito user pools, emails are routed through the same region as the associated user pool.
- If a different email address is used to configure Amazon Simple Email Service (Amazon SES) with Amazon Cognito user pools, that email address is routed through the AWS Region associated with the email address in Amazon SES.
- SMS messages from Amazon Cognito user pools are routed through the same region Amazon SNS unless noted otherwise on Configuring email or phone verification.
- If Amazon Pinpoint analytics are used with Amazon Cognito user pools, the event data is routed to the US East (N. Virginia) Region.

> **Note**
>
> Amazon Pinpoint is available in several AWS Regions in North America, Europe, Asia, and Oceania. Amazon Pinpoint regions include the Amazon Pinpoint API. If a Amazon Pinpoint region is supported by Amazon Cognito, then Amazon Cognito will send events to Amazon Pinpoint projects within the *same* Amazon Pinpoint region. If a region *isn't* supported by Amazon Pinpoint, then Amazon Cognito will *only* support sending events in us-east-1. For Amazon Pinpoint detailed region information, see Amazon Pinpoint endpoints and quotas and Using Amazon Pinpoint analytics with amazon cognito user pools.

Infrastructure security in Amazon Cognito

As a managed service, Amazon Cognito is protected by AWS global network security. For information about AWS security services and how AWS protects infrastructure, see AWS Cloud Security. To design your AWS environment using the best practices for infrastructure security, see Infrastructure Protection in *Security Pillar AWS Well-Architected Framework*.

You use AWS published API calls to access Amazon Cognito through the network. Clients must support the following:

- Transport Layer Security (TLS). We require TLS 1.2 and recommend TLS 1.3.
- Cipher suites with perfect forward secrecy (PFS) such as DHE (Ephemeral Diffie-Hellman) or ECDHE (Elliptic Curve Ephemeral Diffie-Hellman). Most modern systems such as Java 7 and later support these modes.

Configuration and vulnerability analysis in Amazon Cognito user pools

AWS handles basic security tasks like guest operating system (OS) and database patching, firewall configuration, and disaster recovery. These procedures have been reviewed and certified by the appropriate third parties. For more details, see the following resources:

- Compliance validation for Amazon Cognito
- Shared Responsibility Model

AWS managed policies for Amazon Cognito

To add permissions to users, groups, and roles, it is easier to use AWS managed policies than to write policies yourself. It takes time and expertise to create IAM customer managed policies that provide your team with only the permissions they need. To get started quickly, you can use our AWS managed policies. These policies cover common use cases and are available in your AWS account. For more information about AWS managed policies, see AWS managed policies in the *IAM User Guide*.

AWS services maintain and update AWS managed policies. You can't change the permissions in AWS managed policies. Services occasionally add additional permissions to an AWS managed policy to support new features. This type of update affects all identities (users, groups, and roles) where the policy is attached. Services are most likely to update an AWS managed policy when a new feature is launched or when new operations become available. Services do not remove permissions from an AWS managed policy, so policy updates won't break your existing permissions.

Additionally, AWS supports managed policies for job functions that span multiple services. For example, the **ReadOnlyAccess** AWS managed policy provides read-only access to all AWS services and resources. When a service launches a new feature, AWS adds read-only permissions for new operations and resources. For a list and descriptions of job function policies, see AWS managed policies for job functions in the *IAM User Guide*.

AWS managed IAM policies that grant access to Amazon Cognito

- `AmazonCognitoPowerUser` - Permissions for accessing and managing all aspects of your identity pools and user pools. To view the permissions for this policy, see AmazonCognitoPowerUser.
- `AmazonCognitoReadOnly` - Permissions for read-only access to your identity pools and user pools. To view the permissions for this policy, see AmazonCognitoReadOnly.
- `AmazonCognitoDeveloperAuthenticatedIdentities` - Permissions for your authentication system to integrate with Amazon Cognito. To view the permissions for this policy, see AmazonCognitoDeveloperAuthenticatedIdentities.

These policies are maintained by the Amazon Cognito team, so even as new APIs are added, your users continue to have the same level of access.

> **Note**
>
> When you create a new identity pool, you can automatically create new roles for authenticated and guest user access. The administrator who creates your identity pool with new IAM roles must also have IAM permissions to create roles.

Identity pools with unauthenticated guest access apply an additional AWS managed policy as a session policy to unauthenticated users. This AWS managed policy has no intended administrative use. Instead, it limits the scope of permissions that you can apply to guest users in the identity pools enhanced authentication flow. For more information, see IAM roles.

AWS managed IAM policies that Amazon Cognito grants to guest users

- `AmazonCognitoUnAuthedIdentitiesSessionPolicy` - In combination with an inline session policy, limits the permissions that IAM administrators can grant to identity pool guest users. Amazon Cognito automatically applies this policy to guest sessions. For more information, see The AWS managed session policy for guests.

Amazon Cognito updates to AWS managed policies

View details about updates to AWS managed policies for Amazon Cognito since this service began tracking these changes. For automatic alerts about changes to this page, subscribe to the RSS feed on the Amazon Cognito Document history page.

Change	Description	Date
`AmazonCognitoPowerUser` –Change	Amazon Cognito added new actions to permit the use of the AWS End User Messaging SMS API operation DescribeAccountAttributes for Amazon Cognito user pools administrative power users.	February 27, 2025
`AmazonCognitoUnAuthedIdentitiesSessionPolicy` –Change	Amazon Cognito added new actions to permit the use of AWS Key Management Service for unauthenticated (guest) users in identity pools.	October 30, 2024
`AmazonCognitoUnAuthedIdentitiesSessionPolicy` –Change	Amazon Cognito added new actions to permit the use of Amazon Location Service for	August 9, 2024

Change	Description	Date
	unauthenticated (guest) users in identity pools.	
`AmazonCognitoUnAuthedIdentitiesSessionPolicy` –New policy	Added an AWS managed policy for privilege scope-down of guest users in identity pools.	July 14, 2023
`AmazonCognitoPowerUser` and `AmazonCognitoReadOnly` –Change	Added new permissions to allow power users to view and manage associations of AWS WAF web ACLs to Amazon Cognito user pools. Added new permissions to allow read-only users to view associations of AWS WAF web ACLs to Amazon Cognito user pools.	July 19, 2022
`AmazonCognitoPowerUser` –Change	Added a new permission to allow Amazon Cognito to call Amazon Simple Email Service `PutIdentityPolicy` and `ListConfigurationSets` operations. This change allows Amazon Cognito user pools to update Amazon SES sending authorization policies and to apply Amazon SES configuration sets when you configure email sending in your user pool.	November 17, 2021

Change	Description	Date
`AmazonCognitoPowerUser` –Change	Added a new permission to allow Amazon Cognito to call Amazon Simple Notification Service's `GetSMSSandboxAccountStatus` operation. This change allows Amazon Cognito user pools to decide if you need to graduate out of the Amazon Simple Notification Service sandbox in order to send messages to all end users through user pools.	June 1, 2021
Amazon Cognito started tracking changes	Amazon Cognito started tracking changes for its AWS managed policies.	March 1, 2021

Legacy

Important

Amazon Cognito Sync is closed to new customers. If you are starting a new application, use a current alternative such as AWS AppSync and DynamoDB instead. The content in this section is retained for customers with existing Amazon Cognito Sync implementations.

This section collects earlier-generation Amazon Cognito features. Amazon Cognito Sync is closed to new customers, as noted above. The classic hosted UI is an earlier branding version that you can still select per app client, but managed login is the current branding version for new work; to move an app client from the classic hosted UI to managed login, see Migrating from the classic hosted UI to managed login.

Topics

- Amazon Cognito Sync
- Using Amazon Cognito Sync with identity pools
- Customizing hosted UI (classic) branding

Amazon Cognito Sync

Note

Amazon Cognito Sync is no longer open to new customers. For alternatives to Amazon Cognito Sync, please explore AWS AppSync and DynamoDB. Learn more.

Note

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Amazon Cognito Sync is an AWS service and client library that makes it possible to sync application-related user data across devices. Amazon Cognito Sync can synchronize user profile data across mobile devices and the web without using your own backend. The client libraries cache data locally so that your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data. If you set up push sync, you can notify other devices immediately that an update is available.

For information about Amazon Cognito Identity region availability, see AWS Service Region Availability.

To learn more about Amazon Cognito Sync, see the following topics.

Topics

- Amazon Cognito Sync availability change
- Getting started with Amazon Cognito Sync
- Synchronizing data across clients
- Handling event callbacks
- Implementing push synchronization
- Implementing Amazon Cognito Sync streams
- Customizing workflows with Amazon Cognito Events

Amazon Cognito Sync availability change

Amazon Cognito Sync is no longer open to new customers. For alternatives to Amazon Cognito Sync, please explore AWS AppSync and DynamoDB.

This page provides information about Amazon Cognito Sync changes and includes alternatives for Amazon Cognito Sync customers.

AWS AppSync is a serverless service that lets developers create and manage GraphQL APIs. DynamoDB is a key-value store for per-user data such as profiles, preferences, and application settings. Migration from Amazon Cognito Sync requires data transfer and updates to your applications to the different APIs and query patterns by the new service.

If you have additional questions, contact AWS Support.

Frequently asked questions

What does this mean for the service?

Amazon Cognito Sync will no longer be accepting new customers. The service will continue to operate for existing customers but there will be no new feature development.

How will existing customers be impacted?

Existing customers will not experience any disruption to their workloads. You can continue using Amazon Cognito Sync as normal.

What are the recommended alternatives to Amazon Cognito Sync?

- AWS AppSync – For real-time data synchronization across devices with GraphQL APIs.
- DynamoDB – For simple key-value user data storage (profiles, preferences, settings).

How do I migrate my data?

Migration requires updating your data schema, migrating data from the Amazon Cognito Sync store to your chosen alternative, and updating client applications to use the new services.

How can I get help if I have issues?

If you have questions about this, contact AWS Support.

Getting started with Amazon Cognito Sync

Note

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Note

The code examples in this section use the Amazon Cognito Sync client in the archived AWS Mobile SDK (for Android, iOS, Unity, and Xamarin/.NET). That Mobile SDK is no

longer maintained and its packages are not available in current package repositories. The examples remain for existing customers who still integrate against that SDK. For new development, use AWS AppSync.

Amazon Cognito Sync is an AWS service and client library that enable cross-device syncing of application-related user data. You can use it to synchronize user profile data across mobile devices and web applications. The client libraries cache data locally so your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data, and if you set up push sync, notify other devices immediately that an update is available.

Set up an identity pool in Amazon Cognito

Amazon Cognito Sync requires an Amazon Cognito identity pool to provide user identities. Before you use Amazon Cognito Sync you must first set up an identity pool. To create an identity pool and install the SDK, see Getting started with Amazon Cognito identity pools.

Store and sync data

After you have set up your identity pool and installed the SDK, you can start storing and syncing data between devices. For more information, see Synchronizing data across clients.

Synchronizing data across clients

Note

If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

With Amazon Cognito, you can save user data in datasets that contain key-value pairs. Amazon Cognito associates this data with an identity in your identity pool so that your app can access it across logins and devices. To sync this data between the Amazon Cognito service and an end user's devices, invoke the synchronize method. Each dataset can have a maximum size of 1 MB. You can associate up to 20 datasets with an identity.

The Amazon Cognito Sync client creates a local cache for the identity data. When your app reads and writes keys, it communicates with this local cache . This communication guarantees that all changes you make on the device are immediately available on the device, even when you are offline. When the synchronize method is called, changes from the service are pulled to the device, and any local changes are pushed to the service. At this point, the changes are available to other devices to synchronize.

Initializing the Amazon Cognito Sync client

To initialize the Amazon Cognito Sync client, you must first create a credentials provider. The credentials provider acquires temporary AWS credentials to make it possible for your app to access your AWS resources. You also must import the necessary header files. Use the following steps to initialize the Amazon Cognito Sync client.

Android

1. Create a credentials provider, following the instructions in Getting credentials.
2. Import the Amazon Cognito package as follows: `import com.amazonaws.mobileconnectors.cognito.*;`
3. Initialize Amazon Cognito Sync. Pass in the Android app context, the identity pool ID, an AWS Region, and an initialized Amazon Cognito credentials provider as follows:

```
CognitoSyncManager client = new CognitoSyncManager(
    getApplicationContext(),
    Regions.YOUR_REGION,
    credentialsProvider);
```

iOS - Objective-C

1. Create a credentials provider, following the instructions in Getting credentials.
2. Import `AWSCore` and `Cognito`, and initialize `AWSCognito` as follows:

```
#import <AWSiOSSDKv2/AWSCore.h>
#import <AWSCognitoSync/Cognito.h>

AWSCognito *syncClient = [AWSCognito defaultCognito];
```

3. If you're using CocoaPods, replace `<AWSiOSSDKv2/AWSCore.h>` with `AWSCore.h`. Follow the same syntax for the Amazon Cognito import.

iOS - Swift

1. Create a credentials provider, following the instructions in Getting credentials.
2. Import and initialize `AWSCognito` as follows:

```
import AWSCognito
let syncClient = AWSCognito.default()!
```

JavaScript

1. Download the Amazon Cognito Sync Manager for JavaScript.
2. Include the Sync Manager library in your project.
3. Create a credentials provider, following the instructions in Getting credentials.
4. Initialize the Sync Manager as follows:

```
var syncManager = new AWS.CognitoSyncManager();
```

Unity

1. Create an instance of `CognitoAWSCredentials`, following the instructions in Getting credentials.
2. Create an instance of `CognitoSyncManager`. Pass the `CognitoAwsCredentials` object and a `AmazonCognitoSyncConfig`, and include at least the Region set, as follows:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =
 REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Xamarin

1. Create an instance of `CognitoAWSCredentials`, following the instructions in Getting credentials.
2. Create an instance of `CognitoSyncManager`. Pass the `CognitoAwsCredentials` object and a `AmazonCognitoSyncConfig`, and include at least the Region set, as follows:

```
AmazonCognitoSyncConfig clientConfig = new AmazonCognitoSyncConfig { RegionEndpoint =
 REGION };
CognitoSyncManager syncManager = new CognitoSyncManager(credentials, clientConfig);
```

Understanding datasets

Amazon Cognito organizes user profile data into datasets. Each dataset can contain up to 1MB of data in the form of key-value pairs. A dataset is the most granular entity that you can synchronize. Read and write operations performed on a dataset only affect the local store until the synchronize method is invoked. Amazon Cognito identifies a dataset by a unique string. You can create a new dataset or open an existing one as follows.

Android

```
Dataset dataset = client.openOrCreateDataset("datasetname");
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito as follows:

```
dataset.delete();
dataset.synchronize(syncCallback);
```

iOS - Objective-C

```
AWSCognitoDataset *dataset = [syncClient openOrCreateDataset:@"myDataSet"];
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito as follows:

```
[dataset clear];
[dataset synchronize];
```

iOS - Swift

```
let dataset = syncClient.openOrCreateDataset("myDataSet")!
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method as follows: to delete the dataset from Amazon Cognito:

```
dataset.clear()
dataset.synchronize()
```

JavaScript

```
syncManager.openOrCreateDataset('myDatasetName', function(err, dataset) {
   // ...
});
```

Unity

```
string myValue = dataset.Get("myKey");
dataset.Put("myKey", "newValue");
```

To delete a key from a dataset, use `Remove` as follows:

```
dataset.Remove("myKey");
```

Xamarin

```
Dataset dataset = syncManager.OpenOrCreateDataset("myDatasetName");
```

To delete a dataset, first call the method to remove it from local storage, then call the `synchronize` method to delete the dataset from Amazon Cognito as follows:

```
dataset.Delete();
dataset.SynchronizeAsync();
```

Reading and writing data in datasets

Amazon Cognito datasets function as dictionaries, with values accessible by key. You can read, add, or modify keys and values of a dataset just as if the dataset were a dictionary, as shown in the following examples.

Note that values you write to a dataset only affect the local cached copy of the data until you call the synchronize method.

Android

```
String value = dataset.get("myKey");
dataset.put("myKey", "my value");
```

iOS - Objective-C

```
[dataset setString:@"my value" forKey:@"myKey"];
NSString *value = [dataset stringForKey:@"myKey"];
```

iOS - Swift

```
dataset.setString("my value", forKey:"myKey")
let value = dataset.stringForKey("myKey")
```

JavaScript

```
dataset.get('myKey', function(err, value) {
  console.log('myRecord: ' + value);
});

dataset.put('newKey', 'newValue', function(err, record) {
  console.log(record);
});

dataset.remove('oldKey', function(err, record) {
  console.log(success);
});
```

Unity

```
string myValue = dataset.Get("myKey");
dataset.Put("myKey", "newValue");
```

Xamarin

```
//obtain a value
string myValue = dataset.Get("myKey");
```

```
// Create a record in a dataset and synchronize with the server
dataset.OnSyncSuccess += SyncSuccessCallback;
dataset.Put("myKey", "myValue");
dataset.SynchronizeAsync();

void SyncSuccessCallback(object sender, SyncSuccessEventArgs e) {
  // Your handler code here
}
```

Android

To remove keys from a dataset, use the `remove` method as follows:

```
dataset.remove("myKey");
```

iOS - Objective-C

To delete a key from a dataset, use `removeObjectForKey` as follows:

```
[dataset removeObjectForKey:@"myKey"];
```

iOS - Swift

To delete a key from a dataset, use `removeObjectForKey` as follows:

```
dataset.removeObjectForKey("myKey")
```

Unity

To delete a key from a dataset, use `Remove` as follows:

```
dataset.Remove("myKey");
```

Xamarin

You can use `Remove` to delete a key from a dataset:

```
dataset.Remove("myKey");
```

Synchronizing local data with the sync store

Android

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.synchronize(syncCallback);
```

The `synchronize` method receives an implementation of the `SyncCallback` interface, discussed below.

The `synchronizeOnConnectivity()` method attempts to synchronize when connectivity is available. If connectivity is immediately available, `synchronizeOnConnectivity()` behaves like `synchronize()`. Otherwise it monitors for connectivity changes and performs a sync once connectivity is available. If `synchronizeOnConnectivity()`is called multiple times, only the last synchronize request is kept, and only the last callback will fire. If either the dataset or the callback is garbage-collected, this method won't perform a sync, and the callback won't fire.

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

iOS - Objective-C

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

The `synchronize` method is asynchronous and returns an `AWSTask` object to handle the response:

```
[[dataset synchronize] continueWithBlock:^id(AWSTask *task) {
    if (task.isCancelled) {
        // Task cancelled.
    } else if (task.error) {
```

```
        // Error while executing task.
    } else {
        // Task succeeded. The data was saved in the sync store.
    }
    return nil;
}];
```

The `synchronizeOnConnectivity` method attempts to synchronize when the device has connectivity. First, `synchronizeOnConnectivity` checks for connectivity and, if the device is online, immediately invokes synchronize and returns the `AWSTask` object associated with the attempt.

If the device is offline, `synchronizeOnConnectivity` 1) schedules a synchronize for the next time the device comes online and 2) returns an `AWSTask` with a nil result. The scheduled synchronize is only valid for the lifecycle of the dataset object. The data will not be synchronized if the app is exited before connectivity is regained. If you want to be notified when events occur during the scheduled synchronize, you must add observers of the notifications found in `AWSCognito`.

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

iOS - Swift

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

The `synchronize` method is asynchronous and returns an `AWSTask` object to handle the response:

```
dataset.synchronize().continueWith(block: { (task) -> AnyObject? in

            if task.isCancelled {
                // Task cancelled.
            } else if task.error != nil {
                // Error while executing task
            } else {
                // Task succeeded. The data was saved in the sync store.
            }
```

```
            return task
})
```

The `synchronizeOnConnectivity` method attempts to synchronize when the device has connectivity. First, `synchronizeOnConnectivity` checks for connectivity and, if the device is online, immediately invokes `synchronize` and returns the `AWSTask` object associated with the attempt.

If the device is offline, `synchronizeOnConnectivity` 1) schedules a synchronize for the next time the device comes online and 2) returns an `AWSTask` object with a nil result. The scheduled synchronize is only valid for the lifecycle of the dataset object. The data will not be synchronized if the app is exited before connectivity is regained. If you want to be notified when events occur during the scheduled synchronize, you must add observers of the notifications found in `AWSCognito`.

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

JavaScript

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.synchronize();
```

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

Unity

The synchronize method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.Synchronize();
```

Synchronize will run asynchronously and will end up calling one of the several callbacks you can specify in the Dataset.

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

Xamarin

The `synchronize` method compares local cached data to the data stored in the Amazon Cognito Sync store. Remote changes are pulled from the Amazon Cognito Sync store; conflict resolution is invoked if any conflicts occur; and updated values on the device are pushed to the service. To synchronize a dataset, call its `synchronize` method:

```
dataset.SynchronizeAsync();
```

To learn more about dataset synchronization and the different callbacks, see Handling event callbacks.

Handling event callbacks

> **Note**
>
> If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
> It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

As an Amazon Cognito Sync developer, you can implement various callbacks to handle different synchronization events and scenarios. The `SyncCallback` interface in the Android SDK configures notifications about dataset synchronization, including `onSuccess()` when a dataset is successfully downloaded, `onFailure()` when an exception occurs, and `onConflict()` to resolve conflicts between local and remote data.

In the iOS SDK, you can register for similar notifications like `AWSCognitoDidStartSynchronizeNotification` and set handlers like the `AWSCognitoRecordConflictHandler` for conflict resolution. The JavaScript, Unity, and Xamarin platforms have analogous callback mechanisms. When you implement these callbacks, your

application can gracefully handle the various synchronization events and scenarios that can occur when using Amazon Cognito Sync.

Android

SyncCallback Interface

By implementing the `SyncCallback` interface, you can receive notifications on your app about dataset synchronization. Your app can then make active decisions about deleting local data, merging unauthenticated and authenticated profiles, and resolving sync conflicts. You should implement the following methods, which are required by the interface:

- `onSuccess()`
- `onFailure()`
- `onConflict()`
- `onDatasetDeleted()`
- `onDatasetsMerged()`

Note that, if you don't want to specify all the callbacks, you can also use the class `DefaultSyncCallback` which provides default, empty implementations for all of them.

onSuccess

The `onSuccess()` callback is triggered when a dataset is successfully downloaded from the sync store.

```
@Override
public void onSuccess(Dataset dataset, List<Record> newRecords) {
}
```

onFailure

onFailure() is called if an exception occurs during synchronization.

```
@Override
public void onFailure(DataStorageException dse) {
}
```

onConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `onConflict()` method handles conflict resolution. If you don't implement this method, the Amazon Cognito Sync client defaults to using the most recent change.

```
@Override
public boolean onConflict(Dataset dataset, final List<SyncConflict> conflicts) {
    List<Record> resolvedRecords = new ArrayList<Record>();
    for (SyncConflict conflict : conflicts) {
        /* resolved by taking remote records */
        resolvedRecords.add(conflict.resolveWithRemoteRecord());

        /* alternately take the local records */
        // resolvedRecords.add(conflict.resolveWithLocalRecord());

        /* or customer logic, say concatenate strings */
        // String newValue = conflict.getRemoteRecord().getValue()
        //     + conflict.getLocalRecord().getValue();
        // resolvedRecords.add(conflict.resolveWithValue(newValue);
    }
    dataset.resolve(resolvedRecords);

    // return true so that synchronize() is retried after conflicts are resolved
    return true;
}
```

onDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `SyncCallback` interface to confirm whether the local cached copy of the dataset should be deleted too. Implement the `onDatasetDeleted()` method to tell the client SDK what to do with the local data.

```
@Override
public boolean onDatasetDeleted(Dataset dataset, String datasetName) {
    // return true to delete the local copy of the dataset
    return true;
}
```

onDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `onDatasetsMerged()` method:

```
@Override
public boolean onDatasetsMerged(Dataset dataset, List<String> datasetNames) {
    // return false to handle Dataset merge outside the synchronization callback
    return false;
}
```

iOS - Objective-C

Sync Notifications

The Amazon Cognito client will emit a number of `NSNotification` events during a synchronize call. You can register to monitor these notifications via the standard `NSNotificationCenter`:

```
[NSNotificationCenter defaultCenter]
  addObserver:self
  selector:@selector(myNotificationHandler:)
  name:NOTIFICATION_TYPE
  object:nil];
```

Amazon Cognito supports five notification types, listed below.

AWSCognitoDidStartSynchronizeNotification

Called when a synchronize operation is starting. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidEndSynchronizeNotification

Called when a synchronize operation completes (successfully or otherwise). The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidFailToSynchronizeNotification

Called when a synchronize operation fails. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key error which will contain the error that caused the failure.

AWSCognitoDidChangeRemoteValueNotification

Called when local changes are successfully pushed to Amazon Cognito. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that were pushed.

AWSCognitoDidChangeLocalValueFromRemoteNotification

Called when a local value changes due to a synchronize operation. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that changed.

Conflict Resolution Handler

During a sync operation, conflicts may arise if the same key has been modified on the local store and in the sync store. If you haven't set a conflict resolution handler, Amazon Cognito defaults to choosing the most recent update.

By implementing and assigning an AWSCognitoRecordConflictHandler you can alter the default conflict resolution. The AWSCognitoConflict input parameter conflict contains an AWSCognitoRecord object for both the local cached data and for the conflicting record in the sync store. Using the AWSCognitoConflict you can resolve the conflict with the local record: [conflict resolveWithLocalRecord], the remote record: [conflict resolveWithRemoteRecord] or a brand new value: [conflict resolveWithValue:value]. Returning nil from this method prevents synchronization from continuing and the conflicts will be presented again the next time the sync process starts.

You can set the conflict resolution handler at the client level:

```
client.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
 AWSCognitoConflict *conflict) {
    // always choose local changes
    return [conflict resolveWithLocalRecord];
};
```

Or at the dataset level:

```
dataset.conflictHandler = ^AWSCognitoResolvedConflict* (NSString *datasetName,
 AWSCognitoConflict *conflict) {
    // override and always choose remote changes
    return [conflict resolveWithRemoteRecord];
};
```

Dataset Deleted Handler

When a dataset is deleted, the Amazon Cognito client uses the `AWSCognitoDatasetDeletedHandler` to confirm whether the local cached copy of the dataset should be deleted too. If no `AWSCognitoDatasetDeletedHandler` is implemented, the local

data will be purged automatically. Implement an `AWSCognitoDatasetDeletedHandler` if you wish to keep a copy of the local data before wiping, or to keep the local data.

You can set the dataset deleted handler at the client level:

```
client.datasetDeletedHandler = ^BOOL (NSString *datasetName) {
    // make a backup of the data if you choose
    ...
    // delete the local data (default behavior)
    return YES;
};
```

Or at the dataset level:

```
dataset.datasetDeletedHandler = ^BOOL (NSString *datasetName) {
    // override default and keep the local data
    return NO;
};
```

Dataset Merge Handler

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `DatasetMergeHandler`. The handler will receive the name of the root dataset as well as an array of dataset names that are marked as merges of the root dataset.

If no `DatasetMergeHandler` is implemented, these datasets will be ignored, but will continue to use up space in the identity's 20 maximum total datasets.

You can set the dataset merge handler at the client level:

```
client.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {
    // Blindly delete the datasets
    for (NSString *name in datasets) {
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito]
 openOrCreateDataset:name];
       [merged clear];
       [merged synchronize];
    }
};
```

Or at the dataset level:

```
dataset.datasetMergedHandler = ^(NSString *datasetName, NSArray *datasets) {
    // Blindly delete the datasets
    for (NSString *name in datasets) {
        AWSCognitoDataset *merged = [[AWSCognito defaultCognito]
 openOrCreateDataset:name];
        // do something with the data if it differs from existing dataset
        ...
        // now delete it
        [merged clear];
        [merged synchronize];
    }
};
```

iOS - Swift

Sync Notifications

The Amazon Cognito client will emit a number of `NSNotification` events during a synchronize call. You can register to monitor these notifications via the standard `NSNotificationCenter`:

```
NSNotificationCenter.defaultCenter().addObserver(observer: self,
   selector: "myNotificationHandler",
   name:NOTIFICATION_TYPE,
   object:nil)
```

Amazon Cognito supports five notification types, listed below.

AWSCognitoDidStartSynchronizeNotification

Called when a synchronize operation is starting. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidEndSynchronizeNotification

Called when a synchronize operation completes (successfully or otherwise). The `userInfo` will contain the key dataset which is the name of the dataset being synchronized.

AWSCognitoDidFailToSynchronizeNotification

Called when a synchronize operation fails. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key error which will contain the error that caused the failure.

AWSCognitoDidChangeRemoteValueNotification

Called when local changes are successfully pushed to Amazon Cognito. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that were pushed.

AWSCognitoDidChangeLocalValueFromRemoteNotification

Called when a local value changes due to a synchronize operation. The `userInfo` will contain the key dataset which is the name of the dataset being synchronized and the key keys which will contain an NSArray of record keys that changed.

Conflict Resolution Handler

During a sync operation, conflicts may arise if the same key has been modified on the local store and in the sync store. If you haven't set a conflict resolution handler, Amazon Cognito defaults to choosing the most recent update.

By implementing and assigning an `AWSCognitoRecordConflictHandler` you can alter the default conflict resolution. The `AWSCognitoConflict` input parameter conflict contains an `AWSCognitoRecord` object for both the local cached data and for the conflicting record in the sync store. Using the `AWSCognitoConflict` you can resolve the conflict with the local record: [conflict resolveWithLocalRecord], the remote record: [conflict resolveWithRemoteRecord] or a brand new value: [conflict resolveWithValue:value]. Returning nil from this method prevents synchronization from continuing and the conflicts will be presented again the next time the sync process starts.

You can set the conflict resolution handler at the client level:

```
client.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) ->
 AWSCognitoResolvedConflict? in
    return conflict.resolveWithLocalRecord()
}
```

Or at the dataset level:

```
dataset.conflictHandler = {
    (datasetName: String?, conflict: AWSCognitoConflict?) ->
 AWSCognitoResolvedConflict? in
    return conflict.resolveWithLocalRecord()
```

```
}
```

Dataset Deleted Handler

When a dataset is deleted, the Amazon Cognito client uses the `AWSCognitoDatasetDeletedHandler` to confirm whether the local cached copy of the dataset should be deleted too. If no `AWSCognitoDatasetDeletedHandler` is implemented, the local data will be purged automatically. Implement an `AWSCognitoDatasetDeletedHandler` if you wish to keep a copy of the local data before wiping, or to keep the local data.

You can set the dataset deleted handler at the client level:

```
client.datasetDeletedHandler = {
      (datasetName: String!) -> Bool in
      // make a backup of the data if you choose
      ...
      // delete the local data (default behaviour)
      return true
}
```

Or at the dataset level:

```
dataset.datasetDeletedHandler = {
      (datasetName: String!) -> Bool in
      // make a backup of the data if you choose
      ...
      // delete the local data (default behaviour)
      return true
}
```

Dataset merge handler

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `DatasetMergeHandler`. The handler will receive the name of the root dataset as well as an array of dataset names that are marked as merges of the root dataset.

If no `DatasetMergeHandler` is implemented, these datasets will be ignored, but will continue to use up space in the identity's 20 maximum total datasets.

You can set the dataset merge handler at the client level:

```
client.datasetMergedHandler = {
    (datasetName: String!, datasets: [AnyObject]!) -> Void in
    for nameObject in datasets {
        if let name = nameObject as? String {
            let merged = AWSCognito.defaultCognito().openOrCreateDataset(name)
            merged.clear()
            merged.synchronize()
        }
    }
}
```

Or at the dataset level:

```
dataset.datasetMergedHandler = {
    (datasetName: String!, datasets: [AnyObject]!) -> Void in
    for nameObject in datasets {
        if let name = nameObject as? String {
            let merged = AWSCognito.defaultCognito().openOrCreateDataset(name)
            // do something with the data if it differs from existing dataset
              ...
              // now delete it
            merged.clear()
            merged.synchronize()
        }
    }
}
```

JavaScript

Synchronization callbacks

When performing a synchronize() on a dataset, you can optionally specify callbacks to handle each of the following states:

```
dataset.synchronize({

   onSuccess: function(dataset, newRecords) {
      //...
   },

   onFailure: function(err) {
      //...
```

```
   },

   onConflict: function(dataset, conflicts, callback) {
      //...
   },

   onDatasetDeleted: function(dataset, datasetName, callback) {
      //...
   },

  onDatasetMerged: function(dataset, datasetNames, callback) {
      //...
  }

});
```

onSuccess()

The onSuccess() callback is triggered when a dataset is successfully updated from the sync store. If you do not define a callback, the synchronization will succeed silently.

```
onSuccess: function(dataset, newRecords) {
   console.log('Successfully synchronized ' + newRecords.length + ' new records.');
}
```

onFailure()

onFailure() is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
onFailure: function(err) {
   console.log('Synchronization failed.');
   console.log(err);
}
```

onConflict()

Conflicts may arise if the same key has been modified on the local store and in the sync store. The onConflict() method handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
onConflict: function(dataset, conflicts, callback) {
```

```
   var resolved = [];

   for (var i=0; i<conflicts.length; i++) {

      // Take remote version.
      resolved.push(conflicts[i].resolveWithRemoteRecord());

      // Or... take local version.
      // resolved.push(conflicts[i].resolveWithLocalRecord());

      // Or... use custom logic.
      // var newValue = conflicts[i].getRemoteRecord().getValue() +
 conflicts[i].getLocalRecord().getValue();
      // resolved.push(conflicts[i].resovleWithValue(newValue);

   }

   dataset.resolve(resolved, function() {
      return callback(true);
   });

   // Or... callback false to stop the synchronization process.
   // return callback(false);

}
```

onDatasetDeleted()

When a dataset is deleted, the Amazon Cognito client uses the `onDatasetDeleted()` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
onDatasetDeleted: function(dataset, datasetName, callback) {

   // Return true to delete the local copy of the dataset.
   // Return false to handle deleted datasets outside the synchronization callback.

   return callback(true);

}
```

onDatasetMerged()

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the onDatasetsMerged() callback.

```
onDatasetMerged: function(dataset, datasetNames, callback) {

   // Return true to continue the synchronization process.
   // Return false to handle dataset merges outside the synchronization callback.

   return callback(false);

}
```

Unity

After you open or create a dataset, you can set different callbacks to it that will be triggered when you use the Synchronize method. This is the way to register your callbacks to them:

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;
dataset.OnSyncFailure += this.HandleSyncFailure;
dataset.OnSyncConflict = this.HandleSyncConflict;
dataset.OnDatasetMerged = this.HandleDatasetMerged;
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

Note that SyncSuccess and SyncFailure use += instead of = so you can subscribe more than one callback to them.

OnSyncSuccess

The OnSyncSuccess callback is triggered when a dataset is successfully updated from the cloud. If you do not define a callback, the synchronization will succeed silently.

```
private void HandleSyncSuccess(object sender, SyncSuccessEvent e)
{
    // Continue with your game flow, display the loaded data, etc.
}
```

OnSyncFailure

OnSyncFailure is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
private void HandleSyncFailure(object sender, SyncFailureEvent e)
{
    Dataset dataset = sender as Dataset;
    if (dataset.Metadata != null) {
        Debug.Log("Sync failed for dataset : " + dataset.Metadata.DatasetName);
    } else {
        Debug.Log("Sync failed");
    }
    // Handle the error
    Debug.LogException(e.Exception);
}
```

OnSyncConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `OnSyncConflict` callback handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
private bool HandleSyncConflict(Dataset dataset, List < SyncConflict > conflicts)
{
  if (dataset.Metadata != null) {
    Debug.LogWarning("Sync conflict " + dataset.Metadata.DatasetName);
  } else {
    Debug.LogWarning("Sync conflict");
  }
  List < Amazon.CognitoSync.SyncManager.Record > resolvedRecords = new List <
 Amazon.CognitoSync.SyncManager.Record > ();
  foreach(SyncConflict conflictRecord in conflicts) {
    // SyncManager provides the following default conflict resolution methods:
    //      ResolveWithRemoteRecord - overwrites the local with remote records
    //      ResolveWithLocalRecord - overwrites the remote with local records
    //      ResolveWithValue - to implement your own logic
    resolvedRecords.Add(conflictRecord.ResolveWithRemoteRecord());
  }
  // resolves the conflicts in local storage
  dataset.Resolve(resolvedRecords);
  // on return true the synchronize operation continues where it left,
  //      returning false cancels the synchronize operation
  return true;
}
```

OnDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `OnDatasetDeleted` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
private bool HandleDatasetDeleted(Dataset dataset)
  {
      Debug.Log(dataset.Metadata.DatasetName + " Dataset has been deleted");
      // Do clean up if necessary
      // returning true informs the corresponding dataset can be purged in the local
 storage and return false retains the local dataset
      return true;
  }
```

OnDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `OnDatasetsMerged` callback.

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);
        //Lambda function to delete the dataset after fetching it
        EventHandler<SyncSuccessEvent> lambda;
        lambda = (object sender, SyncSuccessEvent e) => {
            ICollection<string> existingValues = localDataset.GetAll().Values;
            ICollection<string> newValues = mergedDataset.GetAll().Values;

            //Implement your merge logic here

            mergedDataset.Delete(); //Delete the dataset locally
            mergedDataset.OnSyncSuccess -= lambda; //We don't want this callback to be
 fired again
            mergedDataset.OnSyncSuccess += (object s2, SyncSuccessEvent e2) => {
                localDataset.Synchronize(); //Continue the sync operation that was
 interrupted by the merge
            };
            mergedDataset.Synchronize(); //Synchronize it as deleted, failing to do so
 will leave us in an inconsistent state
        };
        mergedDataset.OnSyncSuccess += lambda;
        mergedDataset.Synchronize(); //Asnchronously fetch the dataset
```

```
    }

    // returning true allows the Synchronize to continue and false stops it
    return false;
}
```

Xamarin

After you open or create a dataset, you can set different callbacks to it that will be triggered when you use the Synchronize method. This is the way to register your callbacks to them:

```
dataset.OnSyncSuccess += this.HandleSyncSuccess;
dataset.OnSyncFailure += this.HandleSyncFailure;
dataset.OnSyncConflict = this.HandleSyncConflict;
dataset.OnDatasetMerged = this.HandleDatasetMerged;
dataset.OnDatasetDeleted = this.HandleDatasetDeleted;
```

Note that `SyncSuccess` and `SyncFailure` use += instead of = so you can subscribe more than one callback to them.

OnSyncSuccess

The `OnSyncSuccess` callback is triggered when a dataset is successfully updated from the cloud. If you do not define a callback, the synchronization will succeed silently.

```
private void HandleSyncSuccess(object sender, SyncSuccessEventArgs e)
{
    // Continue with your game flow, display the loaded data, etc.
}
```

OnSyncFailure

`OnSyncFailure` is called if an exception occurs during synchronization. If you do not define a callback, the synchronization will fail silently.

```
private void HandleSyncFailure(object sender, SyncFailureEventArgs e)
{
    Dataset dataset = sender as Dataset;
    if (dataset.Metadata != null) {
        Console.WriteLine("Sync failed for dataset : " + dataset.Metadata.DatasetName);
    } else {
```

```
        Console.WriteLine("Sync failed");
    }
}
```

OnSyncConflict

Conflicts may arise if the same key has been modified on the local store and in the sync store. The `OnSyncConflict` callback handles conflict resolution. If you don't implement this method, the synchronization will be aborted when there is a conflict.

```
private bool HandleSyncConflict(Dataset dataset, List < SyncConflict > conflicts)
{
  if (dataset.Metadata != null) {
    Console.WriteLine("Sync conflict " + dataset.Metadata.DatasetName);
  } else {
    Console.WriteLine("Sync conflict");
  }
  List < Amazon.CognitoSync.SyncManager.Record > resolvedRecords = new List <
 Amazon.CognitoSync.SyncManager.Record > ();
  foreach(SyncConflict conflictRecord in conflicts) {
    // SyncManager provides the following default conflict resolution methods:
    //      ResolveWithRemoteRecord - overwrites the local with remote records
    //      ResolveWithLocalRecord - overwrites the remote with local records
    //      ResolveWithValue - to implement your own logic
    resolvedRecords.Add(conflictRecord.ResolveWithRemoteRecord());
  }
  // resolves the conflicts in local storage
  dataset.Resolve(resolvedRecords);
  // on return true the synchronize operation continues where it left,
  //      returning false cancels the synchronize operation
  return true;
}
```

OnDatasetDeleted

When a dataset is deleted, the Amazon Cognito client uses the `OnDatasetDeleted` callback to decide whether the local cached copy of the dataset should be deleted too. By default, the dataset will not be deleted.

```
private bool HandleDatasetDeleted(Dataset dataset)
{
    Console.WriteLine(dataset.Metadata.DatasetName + " Dataset has been deleted");
```

```
    // Do clean up if necessary
    // returning true informs the corresponding dataset can be purged in the local
 storage and return false retains the local dataset
    return true;
}
```

OnDatasetMerged

When two previously unconnected identities are linked together, all of their datasets are merged. Applications are notified of the merge through the `OnDatasetsMerged` callback.

```
public bool HandleDatasetMerged(Dataset localDataset, List<string> mergedDatasetNames)
{
    foreach (string name in mergedDatasetNames)
    {
        Dataset mergedDataset = syncManager.OpenOrCreateDataset(name);

            //Implement your merge logic here

        mergedDataset.OnSyncSuccess += lambda;
        mergedDataset.SynchronizeAsync(); //Asnchronously fetch the dataset
    }

    // returning true allows the Synchronize to continue and false stops it
    return false;
}
```

Implementing push synchronization

> **Note**
>
> If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
> It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Amazon Cognito automatically tracks the association between identity and devices. Using the push synchronization, or push sync, feature, you can ensure that every instance of a given identity is

notified when identity data changes. Push sync ensures that, whenever the sync store data changes for a particular identity, all devices associated with that identity receive a silent push notification informing them of the change.

> **Note**
>
> Push sync is not supported for JavaScript, Unity, or Xamarin.

Before you can use push sync, you must first set up your account for push sync and enable push sync in the Amazon Cognito console.

Create an Amazon Simple Notification Service (Amazon SNS) app

Create and configure an Amazon SNS app for your supported platforms, as described in the SNS Developer Guide.

Enable push sync in the Amazon Cognito console

You can enable push sync via the Amazon Cognito console. From the console home page:

1. Click the name of the identity pool for which you want to enable push sync. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, click **Manage Identity Pools**. The **Federated Identities** page appears.
3. Scroll down and click **Push synchronization** to expand it.
4. In the **Service role** dropdown menu, select the IAM role that grants Cognito permission to send an SNS notification. Click **Create role** to create or modify the roles associated with your identity pool in the AWS IAM Console.
5. Select a platform application, and then click **Save Changes**.
6. Grant SNS Access to Your Application

In the AWS Identity and Access Management console, configure your IAM roles to have full Amazon SNS access, or create a new role that has full Amazon SNS access. The following example role trust policy grants Amazon Cognito Sync a limited ability to assume an IAM role. Amazon Cognito Sync can only assume the role when it does so on behalf of both the identity pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

JSON

```
{
    "Version":"2012-10-17",
    "Statement": [
        {
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-sync.amazonaws.com"
            },
            "Action": "sts:AssumeRole",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceAccount": "123456789012"
                },
                "ArnLike": {
                    "AWS:SourceArn": "arn:aws-eusc:cognito-
identity:eusc-de-east-1:123456789012:identitypool/eusc-de-
east-1:177a950c-2c08-43f0-9983-28727EXAMPLE"
                }
            }
        }
    ]
}
```

To learn more about IAM roles, see Roles (Delegation and Federation).

Use push sync in your app: Android

Your application will need to import the Google Play services. You can download the latest version of the Google Play SDK via the Android SDK manager. Follow the Android documentation on Android Implementation to register your app and receive a registration ID from GCM. Once you have the registration ID, you need to register the device with Amazon Cognito, as shown in the snippet below:

```
String registrationId = "MY_GCM_REGISTRATION_ID";
try {
    client.registerDevice("GCM", registrationId);
} catch (RegistrationFailedException rfe) {
    Log.e(TAG, "Failed to register device for silent sync", rfe);
} catch (AmazonClientException ace) {
```

```
    Log.e(TAG, "An unknown error caused registration for silent sync to fail", ace);
}
```

You can now subscribe a device to receive updates from a particular dataset:

```
Dataset trackedDataset = client.openOrCreateDataset("myDataset");
if (client.isDeviceRegistered()) {
    try {
        trackedDataset.subscribe();
    } catch (SubscribeFailedException sfe) {
        Log.e(TAG, "Failed to subscribe to datasets", sfe);
    } catch (AmazonClientException ace) {
        Log.e(TAG, "An unknown error caused the subscription to fail", ace);
    }
}
```

To stop receiving push notifications from a dataset, simply call the unsubscribe method.
To subscribe to all datasets (or a specific subset) in the `CognitoSyncManager` object, use `subscribeAll()`:

```
if (client.isDeviceRegistered()) {
    try {
        client.subscribeAll();
    } catch (SubscribeFailedException sfe) {
        Log.e(TAG, "Failed to subscribe to datasets", sfe);
    } catch (AmazonClientException ace) {
        Log.e(TAG, "An unknown error caused the subscription to fail", ace);
    }
}
```

In your implementation of the Android BroadcastReceiver object, you can check the latest version of the modified dataset and decide if your app needs to synchronize again:

```
@Override
public void onReceive(Context context, Intent intent) {

    PushSyncUpdate update = client.getPushSyncUpdate(intent);

    // The update has the source (cognito-sync here), identityId of the
    // user, identityPoolId in question, the non-local sync count of the
    // data set and the name of the dataset. All are accessible through
```

```
    // relevant getters.

    String source = update.getSource();
    String identityPoolId = update.getIdentityPoolId();
    String identityId = update.getIdentityId();
    String datasetName = update.getDatasetName();
    long syncCount = update.getSyncCount();

    Dataset dataset = client.openOrCreateDataset(datasetName);

    // need to access last sync count. If sync count is less or equal to
    // last sync count of the dataset, no sync is required.

    long lastSyncCount = dataset.getLastSyncCount();
    if (lastSyncCount < syncCount) {
        dataset.synchronize(new SyncCallback() {
            // ...
        });
    }

}
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.
- `identityId`: The identity ID within the pool.
- `datasetName`: The name of the dataset that was updated. This is available for the sake of the openOrCreateDataset call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Use push sync in your app: iOS - Objective-C

To obtain a device token for your app, follow the Apple documentation on Registering for Remote Notifications. Once you've received the device token as an NSData object from APNs, you'll need to register the device with Amazon Cognito using the `registerDevice:` method of the sync client, as shown below:

```
AWSCognito *syncClient = [AWSCognito defaultCognito];
    [[syncClient registerDevice: devToken] continueWithBlock:^id(AWSTask *task) {
        if(task.error){
            NSLog(@"Unable to registerDevice: %@", task.error);
        } else {
            NSLog(@"Successfully registered device with id: %@", task.result);
        }
        return nil;
      }
    ];
```

In debug mode, your device will register with the APNs sandbox; in release mode, it will register with APNs. To receive updates from a particular dataset, use the `subscribe` method:

```
[[[syncClient openOrCreateDataset:@"MyDataset"] subscribe]
 continueWithBlock:^id(AWSTask *task) {
        if(task.error){
            NSLog(@"Unable to subscribe to dataset: %@", task.error);
        } else {
            NSLog(@"Successfully subscribed to dataset: %@", task.result);
        }
        return nil;
      }
    ];
```

To stop receiving push notifications from a dataset, simply call the `unsubscribe` method:

```
[[[syncClient openOrCreateDataset:@”MyDataset”] unsubscribe]
 continueWithBlock:^id(AWSTask *task) {
        if(task.error){
            NSLog(@"Unable to unsubscribe from dataset: %@", task.error);
        } else {
            NSLog(@"Successfully unsubscribed from dataset: %@", task.result);
        }
        return nil;
      }
    ];
```

To subscribe to all datasets in the `AWSCognito` object, call `subscribeAll`:

```
[[syncClient subscribeAll] continueWithBlock:^id(AWSTask *task) {
```

```
        if(task.error){
            NSLog(@"Unable to subscribe to all datasets: %@", task.error);
        } else {
            NSLog(@"Successfully subscribed to all datasets: %@", task.result);
        }
        return nil;
      }
    ];
```

Before calling `subscribeAll`, be sure to synchronize at least once on each dataset, so that the datasets exist on the server.

To react to push notifications, you need to implement the `didReceiveRemoteNotification` method in your app delegate:

```
- (void)application:(UIApplication *)application didReceiveRemoteNotification:
(NSDictionary *)userInfo
    {
        [[NSNotificationCenter defaultCenter]
 postNotificationName:@"CognitoPushNotification" object:userInfo];
    }
```

If you post a notification using notification handler, you can then respond to the notification elsewhere in the application where you have a handle to your dataset. If you subscribe to the notification like this ...

```
[[NSNotificationCenter defaultCenter] addObserver:self
 selector:@selector(didReceivePushSync:)
    name: :@"CognitoPushNotification" object:nil];
```

...you can act on the notification like this:

```
- (void)didReceivePushSync:(NSNotification*)notification
    {
        NSDictionary * data = [(NSDictionary *)[notification object]
 objectForKey:@"data"];
        NSString * identityId = [data objectForKey:@"identityId"];
        NSString * datasetName = [data objectForKey:@"datasetName"];
        if([self.dataset.name isEqualToString:datasetName] && [self.identityId
 isEqualToString:identityId]){
            [[self.dataset synchronize] continueWithBlock:^id(AWSTask *task) {
                if(!task.error){
```

```
                    NSLog(@"Successfully synced dataset");
                }
                return nil;
            }];
        }
    }
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.
- `identityId`: The identity ID within the pool.
- `datasetName`: The name of the dataset that was updated. This is available for the sake of the `openOrCreateDataset` call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Use push sync in your app: iOS - Swift

To obtain a device token for your app, follow the Apple documentation on Registering for Remote Notifications. Once you've received the device token as an NSData object from APNs, you'll need to register the device with Amazon Cognito using the registerDevice: method of the sync client, as shown below:

```
let syncClient = AWSCognito.default()
syncClient.registerDevice(devToken).continueWith(block: { (task: AWSTask!) ->
 AnyObject! in
    if (task.error != nil) {
        print("Unable to register device: " + task.error.localizedDescription)

    } else {
        print("Successfully registered device with id: \(task.result)")
    }
    return task
})
```

In debug mode, your device will register with the APNs sandbox; in release mode, it will register with APNs. To receive updates from a particular dataset, use the `subscribe` method:

```
syncClient.openOrCreateDataset("MyDataset").subscribe().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to subscribe to dataset: " + task.error.localizedDescription)

  } else {
      print("Successfully subscribed to dataset: \(task.result)")
  }
  return task
})
```

To stop receiving push notifications from a dataset, call the `unsubscribe` method:

```
syncClient.openOrCreateDataset("MyDataset").unsubscribe().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to unsubscribe to dataset: " + task.error.localizedDescription)

  } else {
      print("Successfully unsubscribed to dataset: \(task.result)")
  }
  return task
})
```

To subscribe to all datasets in the `AWSCognito` object, call `subscribeAll`:

```
syncClient.openOrCreateDataset("MyDataset").subscribeAll().continueWith(block: { (task:
 AWSTask!) -> AnyObject! in
  if (task.error != nil) {
      print("Unable to subscribe to all datasets: " + task.error.localizedDescription)

  } else {
      print("Successfully subscribed to all datasets: \(task.result)")
  }
  return task
})
```

Before calling `subscribeAll`, be sure to synchronize at least once on each dataset, so that the datasets exist on the server.

To react to push notifications, you need to implement the `didReceiveRemoteNotification` method in your app delegate:

```
func application(application: UIApplication, didReceiveRemoteNotification userInfo:
 [NSObject : AnyObject],
  fetchCompletionHandler completionHandler: (UIBackgroundFetchResult) -> Void) {

 NSNotificationCenter.defaultCenter().postNotificationName("CognitoPushNotification",
 object: userInfo)
})
```

If you post a notification using notification handler, you can then respond to the notification elsewhere in the application where you have a handle to your dataset. If you subscribe to the notification like this ...

```
NSNotificationCenter.defaultCenter().addObserver(observer:self,
   selector:"didReceivePushSync:",
   name:"CognitoPushNotification",
   object:nil)
```

...you can act on the notification like this:

```
func didReceivePushSync(notification: NSNotification) {
    if let data = (notification.object as! [String: AnyObject])["data"] as? [String:
 AnyObject] {
        let identityId = data["identityId"] as! String
        let datasetName = data["datasetName"] as! String

        if self.dataset.name == datasetName && self.identityId == identityId {
          dataset.synchronize().continueWithBlock {(task) -> AnyObject! in
              if task.error == nil {
                print("Successfully synced dataset")
              }
              return nil
          }
        }
    }
}
```

The following keys are available in the push notification payload:

- `source`: cognito-sync. This can serve as a differentiating factor between notifications.
- `identityPoolId`: The identity pool ID. This can be used for validation or additional information, though it's not integral from the receiver's point of view.

- `identityId`: The identity ID within the pool.
- `datasetName`: The name of the dataset that was updated. This is available for the sake of the `openOrCreateDataset` call.
- `syncCount`: The sync count for the remote dataset. You can use this as a way to make sure that the local dataset is out of date, and that the incoming synchronization is new.

Implementing Amazon Cognito Sync streams

> **Note**
>
> If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
> It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito. Developers can now configure a Kinesis stream to receive events as data is updated and synchronized. Amazon Cognito can push each dataset change to a Kinesis stream you own in real time.

Using Amazon Cognito Streams, you can move all of your Sync data to Kinesis, which can then be streamed to a data warehouse tool such as Amazon Redshift for further analysis. To learn more about Kinesis, see Getting Started Using Amazon Kinesis.

Configuring streams

You can set up Amazon Cognito Streams in the Amazon Cognito console. To enable Amazon Cognito Streams in the Amazon Cognito console, you need to select the Kinesis stream to publish to and an IAM role that grants Amazon Cognito permission to put events in the selected stream.

From the console home page:

1. Click the name of the identity pool for which you want to set up Amazon Cognito Streams. The **Dashboard** page for your identity pool appears.
2. In the top-right corner of the **Dashboard** page, click **Manage Identity Pools**. The Manage Federated Identities page appears.

3. Scroll down and click **Cognito Streams** to expand it.
4. In the **Stream name** dropdown menu, select the name of an existing Kinesis stream. Alternatively, click **Create stream** to create one, entering a stream name and the number of shards. To learn about shards and for help on estimating the number of shards needed for your stream, see the Kinesis Developer Guide.
5. In the **Publish role** dropdown menu, select the IAM role that grants Amazon Cognito permission to publish your stream. Click **Create role** to create or modify the roles associated with your identity pool in the AWS IAM Console.
6. In the **Stream status** dropdown menu, select **Enabled** to enable the stream updates. Click **Save Changes**.

After you've successfully configured Amazon Cognito streams, all subsequent updates to datasets in this identity pool will be sent to the stream.

Stream contents

Each record sent to the stream represents a single synchronization. Here is an example of a record sent to the stream:

```
{
    "identityPoolId": "Pool Id",
    "identityId": "Identity Id",
    "dataSetName": "Dataset Name",
    "operation": "(replace|remove)",
    "kinesisSyncRecords": [
        {
            "key": "Key",
            "value": "Value",
            "syncCount": 1,
            "lastModifiedDate": 1424801824343,
            "deviceLastModifiedDate": 1424801824343,
            "op": "(replace|remove)"
        },
        ...
    ],
    "lastModifiedDate": 1424801824343,
    "kinesisSyncRecordsURL": "S3Url",
    "payloadType": "(S3Url|Inline)",
    "syncCount": 1
}
```

For updates that are larger than the Kinesis maximum payload size of 1 MB, Amazon Cognito incudes a presigned Amazon S3 URL that contains the full contents of the update.

After you have configured Amazon Cognito streams, if you delete the Kinesis stream or change the role trust permission so that Amazon Cognito Sync can no longer assume the role, you turn off the Amazon Cognito streams. You must either recreate the Kinesis stream or fix the role, and then you must turn on the stream again.

Bulk publishing

Once you have configured Amazon Cognito streams, you will be able to execute a bulk publish operation for the existing data in your identity pool. After you initiate a bulk publish operation, either via the console or directly via the API, Amazon Cognito will start publishing this data to the same stream that is receiving your updates.

Amazon Cognito does not guarantee uniqueness of data sent to the stream when using the bulk publish operation. You may receive the same update both as an update as well as part of a bulk publish. Keep this in mind when processing the records from your stream.

To bulk publish all of your streams, follow steps 1-6 under Configuring Streams and then click Start bulk publish. You are limited to one ongoing bulk publish operation at any given time and to one successful bulk publish request every 24 hours.

Customizing workflows with Amazon Cognito Events

> **Note**
>
> If you're new to Amazon Cognito Sync, use AWS AppSync. Like Amazon Cognito Sync, AWS AppSync is a service for synchronizing application data across devices.
> It enables user data like app preferences or game state to be synchronized. It also extends these capabilities by allowing multiple users to synchronize and collaborate in real time on shared data.

Amazon Cognito Events allows you to execute an AWS Lambda function in response to important events in Amazon Cognito. Amazon Cognito raises the Sync Trigger event when a dataset is synchronized. You can use the Sync Trigger event to take an action when a user updates data. The function can evaluate and optionally manipulate the data before it is stored in the cloud and

synchronized to the user's other devices. This is useful to validate data coming from the device before it is synchronized to the user's other devices, or to update other values in the dataset based on incoming data such as issuing an award when a player reaches a new level.

The steps below will guide you through setting up a Lambda function that executes each time a Amazon Cognito Dataset is synchronized.

> **Note**
>
> When using Amazon Cognito events, you can only use the credentials obtained from Amazon Cognito Identity. If you have an associated Lambda function, but you call `UpdateRecords` with AWS account credentials (developer credentials), your Lambda function will not be invoked.

Creating a function in AWS Lambda

To integrate Lambda with Amazon Cognito, you first need to create a function in Lambda. To do so:

Selecting the Lambda Function in Amazon Cognito

1. Open the Lambda console.
2. Click Create a Lambda function.
3. On the Select blueprint screen, search for and select "cognito-sync-trigger."
4. On the Configure event sources screen, leave the Event source type set to "Cognito Sync Trigger" and select your identity pool. Click Next.

> **Note**
>
> When configuring a Amazon Cognito Sync trigger outside of the console, you must add Lambda resource-based permissions to allow Amazon Cognito to invoke the function. You can add this permission from the Lambda console (see Using resource-based policies for AWS Lambda) or by using the Lambda AddPermission operation.
>
> **Example Lambda Resource-Based Policy**
>
> The following AWS Lambda resource-based policy grants Amazon Cognito a limited ability to invoke a Lambda function. Amazon Cognito can only invoke the function on behalf of the identity pool in the `aws:SourceArn` condition and the account in the `aws:SourceAccount` condition.

JSON

```
{
    "Version":"2012-10-17",
    "Id": "default",
    "Statement": [
        {
            "Sid": "lambda-allow-cognito-my-function",
            "Effect": "Allow",
            "Principal": {
                "Service": "cognito-sync.amazonaws.com"
            },
            "Action": "lambda:InvokeFunction",
            "Resource": "arn:aws:lambda:eusc-de-
east-1:111122223333:function:MyFunction",
            "Condition": {
                "StringEquals": {
                    "AWS:SourceAccount": "111122223333"
                },
                "ArnLike": {
                    "AWS:SourceArn": "arn:aws:cognito-
identity:eusc-de-east-1:111122223333:identitypool/eusc-de-
east-1:abcdefg-1234-5678-910a-0e8443553f95"
                }
            }
        }
    ]
}
```

5. On the Configure function screen, enter a name and description for your function. Leave Runtime set to "Node.js." Leave the code unchanged for our example. The default example makes no changes to the data being synced. It only logs the fact that the Amazon Cognito Sync Trigger event occurred. Leave Handler name set to "index.handler." For Role, select an IAM role that grants your code permission to access AWS Lambda. To modify roles, see the IAM console. Leave Advanced settings unchanged. Click Next.
6. On the Review screen, review the details and click Create function. The next page displays your new Lambda function.

Now that you have an appropriate function written in Lambda, you need to choose that function as the handler for the Amazon Cognito Sync Trigger event. The steps below walk you through this process.

From the console home page:

1. Click the name of the identity pool for which you want to set up Amazon Cognito Events. The Dashboard page for your identity pool appears.
2. In the top-right corner of the Dashboard page, click Manage Federated Identities. The Manage Federated Identities page appears.
3. Scroll down and click Cognito Events to expand it.
4. In the Sync Trigger dropdown menu, select the Lambda function that you want to trigger when a Sync event occurs.
5. Click Save Changes.

Now, your Lambda function will be executed each time a dataset is synchronized. The next section explains how you can read and modify the data in your function as it is being synchronized.

Writing a Lambda function for sync triggers

Sync triggers follow the programming pattern that service provider interfaces use. Amazon Cognito provides input to your Lambda function in the following JSON format.

```
{
  "version": 2,
  "eventType": "SyncTrigger",
  "region": "us-east-1",
  "identityPoolId": "identityPoolId",
  "identityId": "identityId",
  "datasetName": "datasetName",
  "datasetRecords": {
    "SampleKey1": {
      "oldValue": "oldValue1",
      "newValue": "newValue1",
      "op": "replace"
    },
    "SampleKey2": {
      "oldValue": "oldValue2",
      "newValue": "newValue2",
      "op": "replace"
```

```
    },..
  }
}
```

Amazon Cognito expects the return value of the function to have the same format as the input.

When you write functions for the Sync Trigger event, observe the following:

- When Amazon Cognito calls your Lambda function during UpdateRecords, the function must respond within 5 seconds. If it doesn't, the Amazon Cognito Sync service generates a `LambdaSocketTimeoutException` exception. You can't increase this timeout value.
- If you get a `LambdaThrottledException` exception, try the sync operation again to update the records.
- Amazon Cognito provides all the records present in the dataset as input to the function.
- Records that the app user updates have the op field set as `replace`. The deleted records have op field set as `remove`.
- You can modify any record, even if the app user doesn't update the record.
- All the fields except the datasetRecords are read-only. Do not change them. If you change these fields, you can't update the records.
- To modify the value of a record, update the value and set the op to `replace`.
- To remove a record, either set the op to `remove`, or set the value to null.
- To add a record, add a new record to the datasetRecords array.
- Amazon Cognito ignores any omitted record in the response when Amazon Cognitoupdates the record.

Sample Lambda function

The following sample Lambda function shows how to access, modify and remove the data.

```
console.log('Loading function');

exports.handler = function(event, context) {
    console.log(JSON.stringify(event, null, 2));

    //Check for the event type
    if (event.eventType === 'SyncTrigger') {
```

```
        //Modify value for a key
        if('SampleKey1' in event.datasetRecords){
            event.datasetRecords.SampleKey1.newValue = 'ModifyValue1';
            event.datasetRecords.SampleKey1.op = 'replace';
        }

        //Remove a key
        if('SampleKey2' in event.datasetRecords){
            event.datasetRecords.SampleKey2.op = 'remove';
        }

        //Add a key
        if(!('SampleKey3' in event.datasetRecords)){
            event.datasetRecords.SampleKey3={'newValue':'ModifyValue3', 'op' :
 'replace'};
        }

    }
    context.done(null, event);
};
```

Using Amazon Cognito Sync with identity pools

Amazon Cognito Sync is an AWS service and client library that makes it possible to sync application-related user data across devices. Amazon Cognito Sync can synchronize user profile data across mobile devices and the web without using your own backend. The client libraries cache data locally so that your app can read and write data regardless of device connectivity status. When the device is online, you can synchronize data. If you set up push sync, you can notify other devices immediately that an update is available.

Managing datasets

If you have implemented Amazon Cognito Sync functionality in your application, the Amazon Cognito identity pools console enables you to manually create and delete datasets and records for individual identities. Any change you make to an identity's dataset or records in the Amazon Cognito identity pools console isn't saved until you select **Synchronize** in the console. The change isn't visible to the end user until the identity calls **Synchronize**. The data being synchronized from other devices for individual identities is visible once you refresh the list datasets page for a particular identity.

Create a dataset for an identity

Amazon Cognito Sync associates a dataset with one identity. You can populate your dataset with identifying information about the user that the identity represents, then sync that information to all of your user's devices.

To add a dataset and dataset records to an identity

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the identity that you want to edit.
4. In **Datasets**, choose **Create dataset**.
5. Enter a **Dataset name** and select **Create dataset**.
6. If you want to add records to your dataset, choose your dataset from identity details. In **Records**, select **Create record**.
7. Enter a **Key** and **Value** for your record. Choose **Confirm**. Repeat to add more records.

Delete a dataset associated with an identity

To delete a dataset and its records from an identity

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **Identity browser** tab.
3. Select the identity that contains the dataset that you want to delete.
4. In **Datasets**, choose the radio button next to the dataset that you want to delete.
5. Select **Delete**. Review your choice and select **Delete** again.

Bulk publish data

Bulk publish can be used to export data already stored in your Amazon Cognito Sync store to a Amazon Kinesis stream. For instructions on how to bulk publish all of your streams, see Implementing Amazon Cognito Sync streams.

Activate push synchronization

Amazon Cognito automatically tracks the association between identity and devices. Using the push sync feature, you can make sure that every instance of a given identity is notified when identity data changes. Push sync makes it so that, whenever the dataset changes for an identity, all devices associated with that identity receive a silent push notification informing them of the change.

You can activate push sync in the Amazon Cognito console.

To activate push synchronization

1. Choose **Identity pools** from the Amazon Cognito console. Select an identity pool.
2. Choose the **Identity pool properties** tab.
3. In **Push synchronization**, select **Edit**
4. Select **Activate push synchronization with your identity pool**.
5. Choose one of the Amazon Simple Notification Service (Amazon SNS) **Platform applications** that you created in the current AWS Region. Amazon Cognito publishes push notifications to your platform application. Select **Create platform application** to navigate to the Amazon SNS console and create a new one.
6. To publish to your platform application, Amazon Cognito assumes an IAM role in your AWS account. Choose to **Create a new IAM role** if you want Amazon Cognito to create a new role for you with basic permissions and a trust relationship with your identity pool. Enter an **IAM role name** to identify your new role, for example `myidentitypool_authenticatedrole`. Select **View policy document** to review the permissions that Amazon Cognito will assign to your new IAM role.
7. You can choose to **Use an existing IAM role** if you already have a role in your AWS account that you want to use. You must configure your IAM role trust policy to include `cognito-identity.amazonaws.com`. Configure your role trust policy to only allow Amazon Cognito to assume the role when it presents evidence that the request originated from an authenticated user in your specific identity pool. For more information, see Role trust and permissions.
8. Select **Save changes**.

Set up Amazon Cognito Streams

Amazon Cognito Streams gives developers control and insight into their data stored in Amazon Cognito Sync. Developers can now configure a Kinesis stream to receive events as data. Amazon

Cognito can push each dataset change to a Kinesis stream you own in real time. For instructions on how to set up Amazon Cognito Streams in the Amazon Cognito console, see Implementing Amazon Cognito Sync streams.

Set up Amazon Cognito Events

Amazon Cognito Events allows you to run an AWS Lambda function in response to important events in Amazon Cognito Sync. Amazon Cognito Sync raises the Sync Trigger event when a dataset is synchronized. You can use the Sync Trigger event to take an action when a user updates data. For instructions on setting up Amazon Cognito Events from the console, see Customizing workflows with Amazon Cognito Events.

To learn more about AWS Lambda, see AWS Lambda.

Customizing hosted UI (classic) branding

You can use the AWS Management Console, or the AWS CLI or API, to specify classic customization settings for the hosted UI. You can upload a custom logo image to be displayed in the app. You can also apply some cascading style sheets (CSS) options to the look and feel of the UI.

You can customize the UI defaults and override individual app clients with specific settings. Amazon Cognito applies the default configuration to every app client that doesn't have client-level settings.

In the Amazon Cognito console and in API requests, the request that sets your UI customization must not exceed 135 KB in size. In rare cases, the sum of request headers, your CSS file, and your logo might exceed 135KB. Amazon Cognito encodes the image file to Base64. This increases the size of a 100 KB image to 130 KB, keeping five KB for request headers and your CSS. If the request is too large, the AWS Management Console or your `SetUICustomization` API request returns a `request parameters too large` error. Adjust your logo image to be no greater than 100KB and your CSS file to be no larger than 3 KB. You can't set CSS and logo customization separately.

> **Note**
>
> To customize your UI, you must set up a domain for your user pool.

Specifying a custom logo in classic branding

Amazon Cognito centers your custom logo above the input fields at the Login endpoint.

Choose a PNG, JPG, or JPEG file that can scale to 350 by 178 pixels for your custom hosted UI logo. Your logo file can be no larger than 100 KB in size, or 130 KB after Amazon Cognito encodes to Base64. To set an `ImageFile` in SetUICustomization in the API, convert your file to a Base64-encoded text string or, in the AWS CLI, provide a file path and let Amazon Cognito encode it for you.

Specifying CSS customizations in classic branding

You can customize the CSS for the hosted app pages, with the following restrictions:

- You can use any of the following CSS class names:
  - `background-customizable`
  - `banner-customizable`
  - `errorMessage-customizable`
  - `idpButton-customizable`
  - `idpButton-customizable:hover`
  - `idpDescription-customizable`
  - `inputField-customizable`
  - `inputField-customizable:focus`
  - `label-customizable`
  - `legalText-customizable`
  - `logo-customizable`
  - `passwordCheck-valid-customizable`
  - `passwordCheck-notValid-customizable`
  - `redirect-customizable`
  - `socialButton-customizable`
  - `submitButton-customizable`
  - `submitButton-customizable:hover`
  - `textDescription-customizable`

- Property values can contain HTML, except for the following values: `@import`, `@supports`, `@page`, or `@media` statements, or Javascript.

You can customize the following CSS properties.

Labels

- **font-weight** is a multiple of 100 from 100 to 900.
- **color** is the text color. Must be a legal CSS color value.

Input fields

- **width** is the width of the containing block as a percentage.
- **height** is the height of the input field in pixels (px).
- **color** is the text color. It can be any standard CSS color value.
- **background-color** is the background color of the input field. It can be any standard CSS color value.
- **border** is a standard CSS border value that specifies the width, transparency, and color of the border of your app window. Width can be any value from 1px to 100px. Transparency can be solid or none. Color can be any standard color value.

Text descriptions

- **padding-top** is the amount of padding above the text description.
- **padding-bottom** is the amount of padding below the text description.
- **display** can be `block` or `inline`.
- **font-size** is the font size for text descriptions.
- **color** is the text color. Must be a legal CSS color value.

Submit button

- **font-size** is the font size of the button text.
- **font-weight** is the font weight of the button text: `bold`, `italic`, or `normal`.
- **margin** is a string of four values indicating the top, right, bottom, and left margin sizes for the button.
- **font-size** is the font size for text descriptions.
- **width** is the width of the button text in percent of the containing block.
- **height** is the height of the button in pixels (px).

- **color** is the button text color. It can be any standard CSS color value.
- **background-color** is the background color of the button. It can be any standard color value.

Banner

- **padding** is a string of four values indicating the top, right, bottom, and left padding sizes for the banner.
- **background-color** is the banner's background color. It can be any standard CSS color value.

Submit button hover

- **color** is the foreground color of the button when you hover over it. It can be any standard CSS color value.
- **background-color** is the background color of the button when you hover over it. It can be any standard CSS color value.

Identity provider button hover

- **color** is the foreground color of the button when you hover over it. It can be any standard CSS color value.
- **background-color** is the background color of the button when you hover over it. It can be any standard CSS color value.

Password check not valid

- **color** is the text color of the `"Password check not valid"` message. It can be any standard CSS color value.

Background

- **background-color** is the background color of the app window. It can be any standard CSS color value.

Error messages

- **margin** is a string of four values indicating the top, right, bottom, and left margin sizes.
- **padding** is the padding size.
- **font-size** is the font size.
- **width** is the width of the error message as a percentage of the containing block.
- **background** is the background color of the error message. It can be any standard CSS color value.
- **border** is a string of three values specifying the width, transparency, and color of the border.
- **color** is the error message text color. It can be any standard CSS color value.

- **box-sizing** is used to indicate to the browser what the sizing properties (width and height) should include.

Identity provider buttons

- **height** is the height of the button in pixels (px).
- **width** is the width of the button text as a percentage of the containing block.
- **text-align** is the text alignment setting. It can be `left`, `right`, or `center`.
- **margin-bottom** is the bottom margin setting.
- **color** is the button text color. It can be any standard CSS color value.
- **background-color** is the background color of the button. It can be any standard CSS color value.
- **border-color** is the color of the button border. It can be any standard CSS color value.

Identity provider descriptions

- **padding-top** is the amount of padding above the description.
- **padding-bottom** is the amount of padding below the description.
- **display** can be `block` or `inline`.
- **font-size** is the font size for descriptions.
- **color** is the text color for IdP section headers for example **Sign in with your corporate ID**. Must be a legal CSS color value.

Legal text

- **color** is the text color. It can be any standard CSS color value.
- **font-size** is the font size.

> **Note**
>
> When you customize **Legal text**, you are customizing the message **We won't post to any of your accounts without asking first** that is displayed under social identity providers in the sign-in page.

Logo

- **max-width** is the maximum width as a percentage of the containing block.
- **max-height** is the maximum height as a percentage of the containing block.

- **background-color** is the color of the background for logs with transparent sections. Must be a legal CSS color value.

Input field focus

- **border-color** is the color of the input field. It can be any standard CSS color value.
- **outline** is the border width of the input field, in pixels.

Social button

- **height** is the height of the button in pixels (px).
- **text-align** is the text alignment setting. It can be `left`, `right`, or `center`.
- **width** is the width of the button text as a percentage of the containing block.
- **margin-bottom** is the bottom margin setting.

Password check valid

- **color** is the text color of the `"Password check valid"` message. It can be any standard CSS color value.

Customizing the hosted UI with classic branding in the AWS Management Console

You can use the AWS Management Console to specify UI customization settings for your app.

> **Note**
>
> You can view the hosted UI with your customizations by constructing the following URL, with the specifics for your user pool, and typing it into a browser: `https://<your_domain>/login?response_type=code&client_id=<your_app_client_id>&redirect_uri=<your_callbac`
> You may have to wait up to one minute to refresh your browser before changes made in the console appear.
> Your domain is shown on the **App integration** tab under **Domain**. Your app client ID and callback URL are shown under **App clients**.

To specify app UI customization settings

1. Sign in to the Amazon Cognito console.

2. In the navigation pane, choose **User Pools**, and choose the user pool you want to edit.
3. Create a domain from the **Domain** tab, or update your existing domain. Under **Branding version**, set your domain to use **Hosted UI (classic)**.
4. Choose the **Managed login** menu.
5. To customize UI settings for all app clients, locate **Style** under **Hosted UI settings** and select **Edit**.
6. To customize UI settings for one app client, go to the **App clients** menu and select the app client you want to modify, then locate **Hosted UI (classic) style** and select **Override**. Select **Edit**.
7. To upload your own logo image file, choose **Choose file** or **Replace current file**.
8. To customize hosted UI CSS, download **CSS template.css** and modify the template with the values you want to customize. Only the keys that are included in the template can be used with the hosted UI. Added CSS keys will not be reflected in your UI. After you have customized the CSS file, choose **Choose file** or **Replace current file** to upload your custom CSS file.

Customizing the hosted UI with classic branding in the user pools API and with the AWS CLI

Use the following commands to specify app UI customization settings for your user pool.

To get the UI customization settings for a user pool's built-in app UI, use the following API operations.

- AWS CLI: `aws cognito-idp get-ui-customization`
- AWS API: GetUICustomization

To set the UI customization settings for a user pool's built-in app UI, use the following API operations.

- AWS CLI from image file: `aws cognito-idp set-ui-customization --user-pool-id <your-user-pool-id> --client-id <your-app-client-id> --image-file fileb://"<path-to-logo-image-file>" --css ".label-customizable{ color: <color>;}"`
- AWS CLI with image encoded as Base64 binary text: `aws cognito-idp set-ui-customization --user-pool-id <your-user-pool-id> --client-id <your-`

```
app-client-id> --image-file <base64-encoded-image-file> --css ".label-
customizable{ color: <color>;}"
```

- AWS API: SetUICustomization

Document history for Amazon Cognito

The following table describes important additions to the documentation for Amazon Cognito. We also make frequent minor updates to the documentation in response to the feedback that you send. To submit feedback, locate the **Feedback** link at the bottom of any page in Amazon Cognito documentation.

Change	Description	Date
Reorganized API and endpoint references	The user pool endpoints and managed login reference is now a top-level section of the guide. The guide no longer links to the Amazon Cognito user pools, Amazon Cognito identity pools, and Amazon Cognito Sync API references. These API references remain available in their respective API reference guides.	October 8, 2026
Status codes in AWS WAF custom block responses	Amazon Cognito now returns a `403` status code in place of `5xx` status codes from AWS WAF custom block responses in managed login and the classic hosted UI.	October 1, 2026
Amazon Cognito Sync is no longer open to new customers.	Amazon Cognito Sync is no longer open to new customers. For alternatives to Amazon Cognito Sync, please explore AWS AppSync and DynamoDB. Learn more.	July 31, 2026

Machine-to-machine authorization with GetClient Token	You can now issue access tokens for machine-to-machine (M2M) authorization with the Amazon Cognito `GetClientToken` API operation, without a user pool domain.	July 29, 2026
Password hash user import	You can now import users with their existing password hashes, enabling immediate sign-in without requiring password resets.	July 13, 2026
Provisioned limit management	You can now manage provisioned limits for your user pool with the Amazon Cognito provisioning APIs.	July 2, 2026
Amazon Cognito Sync will no longer be open to new customers starting July 30, 2026.	Amazon Cognito Sync will no longer be open to new customers starting July 30, 2026. Existing Amazon Cognito Sync customers can continue using the service. For alternatives to Amazon Cognito Sync, please explore AWS AppSync and DynamoDB. Learn more.	June 30, 2026
TLS version selection for custom domains	You can now select the minimum TLS version for your custom domain when you create or update a user pool domain with the Amazon Cognito API.	June 29, 2026

Multi-Region replication for user pools.	You can now set up an additional copy of your user pools in an additional AWS Region.	May 28, 2026
Customer managed KMS keys for user pool encryption at rest.	You can now configure user pool encryption at rest with your own custom KMS key.	May 28, 2026
Updated Lambda trigger retry behavior documentation.	Updated documentation to clarify that Amazon Cognito may retry Lambda function invocations that time out, without specifying a fixed number of retries.	March 23, 2026
Client secret rotation for app clients.	You can now associate up to two secrets per app client for secret rotation. You can also provide your own custom secret value when adding a secret.	January 30, 2026
Inbound Federation Lambda Trigger is available in Amazon Cognito user pools.	You can now use inbound federation lambda trigger to transform federated user attributes during the authentication process with external identity providers.	January 28, 2026
AWS PrivateLink is available in Amazon Cognito identity pools.	You can now send requests to Amazon Cognito identity pool API operations through VPC endpoints.	December 10, 2025

AWS PrivateLink is available in Amazon Cognito user pools.	You can now send requests to Amazon Cognito user pools API operations through VPC endpoints.	November 4, 2025
Amazon Cognito user pools now support resource binding.	Your resource servers can now perform audience verification of access tokens with RFC 8707 resource binding.	October 24, 2025
Added a troubleshooting chapter.	New chapter with solutions to common issues with user pools.	October 24, 2025
Added a new quota for requests for user pool JSON web keys documents.	New quota entry for fetching `.well-known/jwks.json` from user pool OAuth 2.0 authorization servers.	October 13, 2025
Terms and conditions documents in managed login.	You can now add links to your terms & conditions and privacy policy on your managed login sign-up pages.	October 2, 2025
New getting-started example application for identity pools.	You can now set up an example application that showcases the capabilities of identity pools.	September 30, 2025
Updates to requirements for IAM role trust policies for identity pools.	You can now make changes to role trust policies for the Amazon Cognito identity pools service principal only when a condition key scopes the audience for OIDC (web identity) federation to one or more identity pools.	August 1, 2025

Amazon Cognito is now available in the Mexico (Central) AWS Region.	You can now create Amazon Cognito resources in the Mexico (Central) Region.	July 24, 2025
Amazon Cognito is now available in the Asia Pacific (Thailand) AWS Region.	You can now create Amazon Cognito resources in the Asia Pacific (Thailand) Region.	July 24, 2025
AWS WAF web ACLs in managed login.	You can now apply AWS WAF web ACL rules to user pool app clients that have the managed login branding version.	June 24, 2025
Updated Lambda trigger examples.	Updated the example function for custom email and SMS sender Lambda triggers to be compatible with Node.js 22.x. The example is also now more accessible for testing.	May 19, 2025
New `prompt` parameter.	You now have greater control over reauthentication of existing managed login sessions with the `prompt` parameter. You can also pass values for this parameter to third-party providers.	May 15, 2025
Client metadata for M2M requests.	You can now pass client metadata in client credentia ls, or machine-to-machine (M2M), requests. Amazon Cognito passes M2M client metadata to the pre token generation Lambda trigger.	April 29, 2025

Refresh token rotation.	You can now get new refresh tokens and invalidate the originals in refresh-token requests.	April 22, 2025
Amazon Cognito is now available in the Asia Pacific (Malaysia) AWS Region.	You can now create Amazon Cognito resources in the Asia Pacific (Malaysia) Region.	March 7, 2025
Access token customization for machine identities.	The pre token generation Lambda trigger now has a version three event that modifies access token claims and scopes in client-cr edentials grants for machine-to-machine (M2M) authoriza tion.	March 3, 2025
Updated information about `AmazonCognitoPower User` AWS managed policy.	Added an AWS End User Messaging SMS operation in the AWS managed policy for Amazon Cognito user pools power users.	February 27, 2025
Updated overview of OpenID Connect (OIDC) integration.	Added a diagram that illustrat es how Amazon Cognito authenticates with OIDC identity providers.	February 25, 2025
Added information about MFA logic.	Added a diagram that illustrat es how Amazon Cognito applies your user pool multi-factor authentication (MFA) settings to users at runtime.	February 25, 2025

Added Amazon Cognito user pools security best practices.	Added a page about securing secrets and otherwise following security best practices in user pool configuration.	February 25, 2025
Updates to getting-started resources for user pools.	The getting started experienc e with Amazon Cognito user pools has a new console design and application options.	November 21, 2024
New pricing model with feature plans.	Updated the billing model for user pools. Advanced security features are now threat protection. Component s in the advanced security features license are now in the Essentials and Plus feature plans.	November 21, 2024
New managed login feature.	Launched managed login, an update to the hosted UI.	November 21, 2024
A new authentication method and new authentication flows.	You can now sign in to Amazon Cognito user pools with passkeys and one-time passwords.	November 21, 2024
Updated information about `AmazonCognitoUnAut hedIdentitiesSessi onPolicy .`	Moved AWS Key Management Service operations in the AWS managed policy for scope-down of unauthenticated identities from inline policy to AWS managed policy.	November 1, 2024

Added `login_hint` parameter.	You can now add a username hint to authorization requests for the hosted UI, OIDC IdPs, and Google IdPs.	October 3, 2024
New advanced security features for email MFA.	You can now send multi-factor authentication (MFA) codes by email message with advanced security features.	September 12, 2024
New content and page changes.	Modified titles, removed unneeded content, added scenario-based intros, moved user pools OIDC & hosted UI endpoints reference to user pools section.	September 9, 2024
Updated information about `AmazonCognitoUnAuthedIdentitiesSessionPolicy`.	The AWS managed policy for scope-down of unauthenticated identities in identity pools now permits Amazon Location Service.	August 9, 2024
New threat prevention for custom authentication with Lambda triggers and enhanced threat detection.	You can now analyze custom authentication sign-in with threat protection and apply adaptive authentication responses. Threat protection also now analyzes sign-in traffic for impossible geographical distance between attempts.	August 8, 2024

New advanced security features for password reuse prevention and user-activity log export.	You can now export user activity logs and set a password-history policy with advanced security features in Amazon Cognito user pools.	August 6, 2024
Amazon Cognito is now available in the Canada West (Calgary) and Asia Pacific (Hong Kong) AWS Regions.	You can now create Amazon Cognito resources in the Canada West (Calgary)and Asia Pacific (Hong Kong) Regions.	July 9, 2024
Improved description of application behavior for advanced security	Updated information about device context data for advanced security adaptive authentication.	June 10, 2024
Added support for complex objects in pre token Lambda trigger	You can now add arrays and JSON objects to ID and access token claims.	May 30, 2024
Updated information about Verified Permissions and Amazon Cognito.	Amazon Verified Permissions now has more direct integrati on with Amazon Cognito.	May 15, 2024
Multi-Region Amazon SES verified identities.	In some AWS Regions without Amazon SES, Amazon Cognito user pools load balance email between two remote Regions.	May 10, 2024
Added information about M2M authorization and managing costs.	Learn how to use client credentials grants for machine-to-machine (M2M) use cases with Amazon Cognito user pools.	May 9, 2024

Amazon Cognito is now available in the Europe (Spain) and Asia Pacific (Hyderabad) AWS Regions.	You can now create Amazon Cognito resources in the Europe (Spain) and Asia Pacific (Hyderabad) Regions.	April 15, 2024
Amazon Cognito is now available in the Asia Pacific (Melbourne) AWS Region.	You can now create Amazon Cognito resources in the Asia Pacific (Melbourne) Region.	April 4, 2024
Added an example Android app in Flutter for Amazon Cognito user pools.	You can build a starter mobile app for Amazon Cognito from an example Flutter applicati on on GitHub.	April 4, 2024
New getting-started content	Expanded content for getting started, common scenarios, multi-tenant best practices, and accessing resources after sign-in.	April 1, 2024
Amazon Cognito is now available in the Europe (Zurich) AWS Region.	You can now create Amazon Cognito resources in the Europe (Zurich) Region.	March 14, 2024
Amazon Cognito is now available in the Middle East (UAE) AWS Region.	You can now create Amazon Cognito resources in the Middle East (UAE) Region.	March 8, 2024
New SAML features and improved content.	You can now sign SAML requests, encrypt SAML responses, and set up IdP-initi ated SAML SSO.	February 1, 2024
Quota increases available.	You can now purchase additional capacity for Amazon Cognito request-rate quotas.	January 25, 2024

Amazon Cognito identity pools support request rates in Service Quotas.	You can now monitor requests-per-second (RPS) quotas for Amazon Cognito identity pools and request increase in the Service Quotas console.	December 19, 2023
Added a new feature for customization of the contents of access tokens.	You can now add, modify, and remove claims and scopes in user pool access tokens.	December 12, 2023
Improved content about app clients and OAuth scopes.	Clarity edits and correctio ns to Application-specific settings with app clients and Scopes, M2M, and resource servers. Removed legacy console instructions.	November 14, 2023
Improved content about devices and device authentic ation.	New content about the use of device keys and device SRP authentication.	October 18, 2023
Updated AWS Management Console guidance.	Removed user pools console reference and redistributed topics within related subjects, and added guidance to tab-based organization in Amazon Cognito console.	August 30, 2023
De-emphasized direct access to LOGIN endpoint.	Added a visual overview of the user pool Login endpoint and emphasized starting authentication with Authorize endpoint.	August 30, 2023

Amazon Cognito is now available in the Asia Pacific (Osaka) and Israel (Tel Aviv) AWS Regions.	You can now create Amazon Cognito resources in the Asia Pacific (Osaka) and Israel (Tel Aviv) Regions.	August 30, 2023
Introduced information about authorization for Amazon Cognito with Amazon Verified Permissions.	In your app, you can invoke the Verified Permissions API to produce access decisions from a central authority.	August 1, 2023
Added a new feature for logging user pool detailed user activity to Amazon CloudWatch Logs.	You can now log email and SMS message delivery errors to CloudWatch log groups.	August 1, 2023
Updated information about AWS managed policy for identity pool guest users.	Permissions scope-down for identity pool guest users now includes both an inline session policy and an AWS managed session policy.	May 16, 2023
Content improvement and new console instructions for Amazon Cognito identity pools.	Added new console walkthroughs to reflect the new console experience, improved code integration details for identity pools.	May 16, 2023
Additions and improvements to service homepage and user pools homepage.	Updated overview pages for Amazon Cognito and user pools.	May 16, 2023
General improvements to user pool token documentation.	Updated example tokens, added new information about verifying tokens.	February 16, 2023

You can now log Amazon Cognito identity pools data events in AWS CloudTrail.	CloudTrail supports the selection of Amazon Cognito identity pools high-volume API operations in trails that log data events.	February 15, 2023
Updated Lambda trigger examples and descriptions.	Lambda trigger examples are updated to JavaScrip t version 3. You can now directly correlate Lambda triggers to API actions.	January 31, 2023
Amazon Cognito identity pools apply an AWS managed policy to unauthenticated sessions.	Identity pool users who authenticate using the enhanced flow now have an additional AWS managed policy applied to their session.	January 31, 2023
Added code examples.	This guide now includes example code for your Amazon Cognito app in a variety of programming langages.	January 23, 2023
Added information about API models and authentication with Amazon Cognito user pools.	Amazon Cognito user pools have multiple API interface s and formats for request authorization.	December 15, 2022
Amazon Cognito is now available in the Europe (Milan) AWS Region.	You can now create Amazon Cognito user pools in the Europe (Milan) Region.	December 6, 2022
Added information about user pool deletion protection.	When you create a new user pool with the AWS Management Console, it's now protected against deletion by default.	October 20, 2022

Added a user guide for the hosted UI, and informati on about TOTP MFA in the hosted UI.	Your users can now register a TOTP MFA device in the Amazon Cognito hosted UI. You can now preview the default hosted UI.	September 8, 2022
Added information about AWS WAF and Amazon Cognito.	You can now associate a AWS WAF web ACL with a Amazon Cognito user pool.	August 3, 2022
Added more example AWS CloudTrail events.	Amazon Cognito now logs federation and hosted UI requests to your trail.	June 15, 2022
Added information about two-step attribute verificat ion.	You can now choose whether your user must verify a new email address or phone number before they can sign in with it.	June 9, 2022
Updated federation documentation. New IP address propagation feature.	Updated walkthroughs for setting up user pool social IdPs. Added information about federated user profiles and attribute mapping. Added new information about device fingerprints for advanced security.	May 31, 2022
Sign in federated users without interaction with the hosted UI	Added a new page about how to bookmark applications so that Amazon Cognito silently directs users to federated sign-in.	May 29, 2022

In-Region SMS and email messaging for Amazon Cognito user pools	You can now use Amazon Simple Notification Service for SMS messages and Amazon Simple Email Service for email messages in the same AWS Region as your user pool.	March 14, 2022
Updates to quotas page	Added and clarified resource and request-rate quotas.	January 10, 2022
New Amazon Cognito user pools console experience	Updated instructions to create and manage user pools in the updated Amazon Cognito console.	November 18, 2021
RevokeToken API and Revocation Endpoint	You can use the RevokeToken operation to revoke a refresh token for a user.	June 10, 2021
Multi-tenant best practices	Added best practices for multi-tenant applications.	March 4, 2021
Attributes for access control	Amazon Cognito Identity Pools provide attributes for access control (AFAC) as a way for customers to grant users access to AWS resources . Authorization can be done based on users' attributes from the identity provider which they used to federate with Amazon Cognito.	January 15, 2021

Custom SMS Sender Lambda Trigger and Custom Email Sender Lambda Trigger	The Custom SMS Sender Lambda Trigger and Custom Email Sender Lambda Trigger allow you to enable a third-party provider to send email and SMS notifications to your users from within your Lambda function code.	November 30, 2020
Amazon Cognito token updates	Updated expiration informati on was added to Access, ID, and Refresh tokens.	October 29, 2020
Amazon Cognito Service Quotas	Service Quotas are available for Amazon Cognito category quotas. You can use the Service Quotas console to view quota usage, request a quota increase, and create CloudWatch alarms to monitor your quota usage. As part of this change the Available CloudWatch Metrics for Amazon Cognito User Pools section was updated to reflect the new informati on. The new section name is: Tracking quotas and usage in CloudWatch and Service Quotas	October 29, 2020
Amazon Cognito quota categorization	Quota categories are available to help you monitor quota usage and request an increase. The quotas are grouped into categories based on common use cases.	August 17, 2020

Amazon Cognito supported in US AWS GovCLoud	Amazon Cognito is now supported in the AWS GovCloud (US) Region.	May 13, 2020
Amazon Cognito Pinpoint document updates	New service-linked role was added. Instructions were updated on "Using Amazon Pinpoint Analytics with Amazon Cognito User Pools".	May 13, 2020
New Amazon Cognito dedicated security chapter	The Security chapter can help your organization get in-depth information about both the built-in and the configurable security of AWS services. Our new chapters provide information about the security of the cloud and in the cloud.	April 30, 2020
Amazon Cognito Identity Pools now supports Sign in with Apple	Sign in with Apple is available in all regions where Amazon Cognito operates, except cn-north-1 region.	April 7, 2020
New Facebook API Versioning	Added version selection to Facebook API.	April 3, 2020
Username case insensitivity update	Added recommendation about enabling username case insensitivity before creating a user pool.	February 11, 2020

New information about AWS Amplify	Added information about integrating Amazon Cognito with your web or mobile app by using AWS Amplify SDKs and libraries. Removed information about using the Amazon Cognito SDKs that preceded AWS Amplify.	November 22, 2019
New attribute for user pool triggers	Amazon Cognito now includes a `clientMetadata` parameter in the event information that it passes to the AWS Lambda functions for most user pool triggers. You can use this parameter to enhance your custom authentication workflow with additional data.	October 4, 2019
Updated limit	The throttling limit for the ListUsers API action is updated.	June 25, 2019
New limit	The soft limits for user pools now include a limit for the number of users.	June 17, 2019
Amazon SES email settings for Amazon Cognito user pools	You can configure a user pool so that Amazon Cognito emails your users by using your Amazon SES configura tion. This setting allows Amazon Cognito to send email with a higher delivery volume than is otherwise possible.	April 8, 2019

Tagging support	Added information about tagging Amazon Cognito resources.	March 26, 2019
Change the certificate for a custom domain	If you use a custom domain to host the Amazon Cognito hosted UI, you can change the SSL certificate for this domain as needed.	December 19, 2018
New limit	A new limit is added for the maximum number of groups that each user can belong to.	December 14, 2018
Updated limits	The soft limits for user pools are updated.	December 11, 2018
Documentation update for verifying email addresses and phone numbers	Added information about configuring your user pool to require email or phone verification when a user signs up in your app.	November 20, 2018
Documentation update for testing emails	Added guidance for initiating emails from Amazon Cognito while you test your app.	November 13, 2018
Amazon Cognito Advanced Security	Added new security features to enable developers to protect their apps and users from malicious bots, secure user accounts against compromised credentials, and automatically adjust the challenges required to sign in based on the calculated risk of the sign in attempt.	June 14, 2018

Custom Domains for Amazon Cognito Hosted UI	Allow developers to use their own fully custom domain for the hosted UI in Amazon Cognito User Pools.	June 4, 2018
Amazon Cognito User Pools OIDC Identity Provider	Added user pool sign-in through an OpenID Connect (OIDC) identity provider such as Salesforce or Ping Identity.	May 17, 2018
Amazon Cognito Lambda Migration Trigger	Added pages covering the Lambda Migration Trigger feature	April 8, 2018
Amazon Cognito Developer Guide Update	Added top level "What is Amazon Cognito" and "Getting Started with Amazon Cognito". Also added common scenarios and reorganized the user pools TOC. Added a new "Getting Started with Amazon Cognito user pools" section.	April 6, 2018
Amazon Cognito Advanced Security Beta	Added new security features to enable developers to protect their apps and users from malicious bots, secure user accounts against credentials in the wild that have been compromised elsewhere on the internet, and automatically adjust the challenges required to sign in based on the calculated risk of the sign in attempt.	November 28, 2017

Amazon Pinpoint integration	Added the ability to use Amazon Pinpoint to provide analytics for your Amazon Cognito User Pools apps and to enrich the user data for Amazon Pinpoint campaigns.	September 26, 2017
Federation and built-in app UI features of Amazon Cognito user pools	Added the ability to allow your users to sign in to your user pool through Facebook, Google, Login with Amazon, or a SAML identity provider. Added a customizable built-in app UI and OAuth 2.0 support with custom claims.	August 10, 2017
HIPAA and PCI compliance-related feature changes	Added the ability to allow your users to use a phone number or email address as their user name.	July 6, 2017
User groups and role-based access control features	Added administrative capability to create and manage user groups. Administrators can assign IAM roles to users based on group membership and administr ator-created rules.	December 15, 2016
Documentation update	Updated examples that show how to use AWS Lambda triggers with user pools.	November 27, 2016
Documentation update	Updated iOS code examples.	November 18, 2016
Documentation update	Added information about confirmation flow for user accounts.	November 9, 2016

Create user accounts feature	Added administrative capability to create user accounts through the Amazon Cognito console and the API.	October 6, 2016
User import feature	Added bulk import capabilit y for Cognito User Pools. Use this feature to migrate users from your existing identity provider to an Amazon Cognito user pool.	September 1, 2016
General availability of Cognito User Pools	Added the Cognito User Pools feature. Use this feature to create and maintain a user directory and add sign-up and sign-in to your mobile app or web application using user pools.	July 28, 2016
SAML support	Added support for authentic ation with identity providers through Security Assertion Markup Language 2.0 (SAML 2.0).	June 23, 2016
CloudTrail integration	Added integration with AWS CloudTrail.	February 18, 2016
Integration of events with Lambda	Enables you to execute an AWS Lambda function in response to important events in Amazon Cognito.	April 9, 2015
Data stream to Amazon Kinesis	Provides control and insight into your data streams.	March 4, 2015

OpenID Connect support	Enables support for OpenID Connect providers.	November 23, 2014
Push synchronization	Enables support for silent push synchronization.	November 6, 2014
Developer-authenticated identities support added	Enables developers who own their own authentication and identity management systems to be treated as an identity provider in Amazon Cognito.	September 29, 2014
Amazon Cognito general availability		July 10, 2014