aws

Developer Guide

AWS SDK for Go v2

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

AWS SDK for Go v2: 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 the AWS SDK for Go v2? 1
Maintenance and support for SDK major versions 1
Get started 2
Get an Amazon Account 2
Install the AWS SDK for Go v2 2
Get your AWS access keys 3
To get your access key ID and secret access key. 3
Related topics 4
Invoke an Operation 4
Configure the SDK 6
Loading AWS Shared Configuration Files 6
Specifying the AWS Region 7
Configure Region with Environment Variable 7
Specify Region Programmatically 7
Specifying Credentials 7
IAM Roles for Tasks 9
IAM Roles for Amazon EC2 Instances 9
Shared Credentials and Configuration 9
Environment Variables 12
Specify Credentials Programmatically 13
Authentication 17
Definitions 17
Auth scheme resolution workflow 20
Natively-supported `AuthSchemes` 20
Client Endpoints 23
Customization 23
V2: `EndpointResolverV2` + `BaseEndpoint` 24
V1: `EndpointResolver` 28
Migration 30
HTTP Client 34
Overriding During Configuration Loading 34
Timeout 35
Dialer 35
Transport 36

Interceptors ... 39
Interceptors vs. middleware ... 39
Available interceptor hooks ... 40
Interceptor registration ... 41
Global interceptor configuration ... 41
Logging ... 42
Logger ... 42
ClientLogMode ... 43
Traces and metrics ... 43
Tracing ... 44
Metrics ... 44
Retries and Timeouts ... 47
Standard Retryer ... 47
NopRetryer ... 48
Customizing Behavior ... 48
Timeouts ... 52
Using the SDK ... 54
Constructing a Service Client ... 54
NewFromConfig ... 54
New ... 55
Calling Service Operations ... 57
Passing Parameters to a Service Operation ... 57
Overriding Client Options For Operation Call ... 58
Handling Operation Responses ... 59
Concurrently Using Service Clients ... 61
Using Operation Paginators ... 63
Using Waiters ... 64
Overriding waiter configuration ... 66
Advanced waiter configuration overrides ... 67
Handling Errors in the SDK ... 68
Logging Errors ... 69
Service Client Errors ... 69
Retrieving Request Identifiers ... 71
Use AWS services ... 73
Amazon S3 checksums ... 73
Upload an object ... 74

Download an object ... 77
SDK Utilities ... 78
Amazon RDS Utilities ... 78
IAM Authentication ... 78
Amazon CloudFront Utilities ... 79
Amazon CloudFront URL Signer ... 79
Amazon EC2 Instance Metadata Service ... 80
Amazon S3 Utilities ... 81
Amazon S3 Transfer Managers ... 81
Unseekable Streaming Input ... 91
Middleware ... 93
Writing a Custom Middleware ... 94
Attaching Middleware to All Clients ... 96
Attaching Middleware to a Specific Operation ... 96
Passing Metadata Down the Stack ... 97
Metadata Provided by the SDK ... 98
Passing Metadata Up the Stack ... 99
Frequently Asked Questions ... 101
How do I configure my SDK's HTTP client? Are there any guidelines or best practices? ... 101
How should I configure operation timeouts? ... 101
Requests made by the SDK are timing out or taking too long, how do I fix this? ... 101
How do I fix a `read: connection reset` error? ... 102
Why am I getting "invalid signature" errors when using an HTTP proxy with the SDK? ... 103
Timing SDK operations ... 103
Unit Tests ... 110
Mocking Client Operations ... 110
Mocking Paginators ... 112
Code examples ... 115
API Gateway ... 116
AWS community contributions ... 116
Aurora ... 117
Get started ... 117
Basics ... 119
Actions ... 137
Amazon Bedrock ... 156
Get started ... 117

Actions ... 137
Amazon Bedrock Runtime ... 160
Get started ... 117
Amazon Nova ... 162
Amazon Titan Image Generator ... 163
Anthropic Claude ... 166
CloudFormation ... 167
Actions ... 137
CloudWatch Logs ... 169
Actions ... 137
Amazon Cognito Identity Provider ... 171
Get started ... 117
Actions ... 137
Scenarios ... 187
Amazon DocumentDB ... 253
Serverless examples ... 254
DynamoDB ... 255
Basics ... 119
Actions ... 137
Scenarios ... 187
Serverless examples ... 254
AWS community contributions ... 116
IAM ... 328
Get started ... 117
Basics ... 119
Actions ... 137
Kinesis ... 388
Serverless examples ... 254
Lambda ... 391
Get started ... 117
Basics ... 119
Actions ... 137
Scenarios ... 187
Serverless examples ... 254
AWS community contributions ... 116
Amazon MSK ... 503

Serverless examples ... 254
Partner Central ... 504
Actions ... 137
Amazon RDS ... 507
Get started ... 117
Basics ... 119
Actions ... 137
Serverless examples ... 254
Amazon Redshift ... 543
Get started ... 117
Basics ... 119
Actions ... 137
Amazon S3 ... 561
Get started ... 117
Basics ... 119
Actions ... 137
Scenarios ... 187
Serverless examples ... 254
Amazon SNS ... 650
Get started ... 117
Actions ... 137
Scenarios ... 187
Serverless examples ... 254
Amazon SQS ... 678
Get started ... 117
Actions ... 137
Scenarios ... 187
Serverless examples ... 254
Migrate to v2 ... 711
Minimum Go Version ... 711
Modularization ... 711
Configuration Loading ... 712
Examples ... 25
Mocking and `*iface` ... 715
Credentials and Credential Providers ... 716
Static Credentials ... 13

Amazon EC2 IAM Role Credentials ... 718
Endpoint Credentials ... 719
Process Credentials ... 719
AWS Security Token Service Credentials ... 720
Service Clients ... 722
Client Construction ... 723
Endpoints ... 725
Authentication ... 725
Invoking API Operations ... 726
Service Data Types ... 727
Pointer Parameters ... 727
Errors Types ... 728
Paginators ... 730
Waiters ... 731
Presigned Requests ... 731
Request customization ... 733
Operation input/output ... 733
HTTP request/response ... 738
Handler phases ... 740
Features ... 742
Amazon EC2 Instance Metadata Service ... 742
Amazon S3 Transfer Manager ... 743
Amazon CloudFront Signing Utilities ... 743
Amazon S3 Encryption Client ... 743
Service Customizations Changes ... 744
Amazon S3 ... 744
Security ... 746
Data protection ... 746
Compliance validation ... 747
Resilience ... 748
Enforcing a minimum TLS version ... 748
Setting the TLS version ... 748
Document history ... 750

What is the AWS SDK for Go v2?

The AWS SDK for Go v2 provides APIs and utilities that developers can use to build Go applications that use AWS services, such as Amazon Elastic Compute Cloud (Amazon EC2) and Amazon Simple Storage Service (Amazon S3).

The SDK removes the complexity of coding directly against a web service interface. It hides a lot of the lower-level plumbing, such as authentication, request retries, and error handling.

The SDK also includes helpful utilities. For example, the Amazon S3 download and upload manager can automatically break up large objects into multiple parts and transfer them in parallel.

Use the AWS SDK for Go Developer Guide to help you install, configure, and use the SDK. This guide provides configuration information, sample code, and an introduction to the SDK utilities.

Maintenance and support for SDK major versions

For information about maintenance and support for SDK major versions and their underlying dependencies, see the following in the AWS SDKs and Tools Reference Guide:

- AWS SDKs and tools maintenance policy
- AWS SDKs and tools version support matrix

Get started with AWS SDK for Go

The AWS SDK for Go requires a minimum version of Go 1.23. You can view your current version of Go by running the following command:

```
go version
```

For information about installing or upgrading your version of Go, see Download and install in the Go documentation.

Get an Amazon Account

Before you can use the AWS SDK for Go v2, you must have an Amazon account. See Create an AWS account for details.

Install the AWS SDK for Go v2

The AWS SDK for Go v2 uses Go modules, which was a feature introduced in Go 1.11. Initialize your local project by running the following Go command.

```
go mod init example
```

After initializing your Go module project, you will be able to retrieve the SDK and its required dependencies using the `go get` command. These dependencies will be recorded in the `go.mod` file which was created by the previous command.

The following commands show how to retrieve the standard set of SDK modules to use in your application.

```
go get github.com/aws/aws-sdk-go-v2
go get github.com/aws/aws-sdk-go-v2/config
```

This will retrieve the core SDK module, and the config module which is used for loading the AWS shared configuration.

Next you can install one or more AWS service API clients required by your application. All API clients are located under `github.com/aws/aws-sdk-go-v2/service` import hierarchy. A

complete set of currently supported API clients can be found here. To install a service client, execute the following command to retrieve the module and record the dependency in your `go.mod` file. In this example we retrieve the Amazon S3 API client.

```
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Get your AWS access keys

Access keys consist of an access key ID and secret access key, which are used to sign programmatic requests that you make to AWS. If you don't have access keys, you can create them by using the AWS Management Console. We recommend that you use IAM access keys instead of AWS root account access keys. IAM lets you securely control access to AWS services and resources in your AWS account.

> **Note**
>
> To create access keys, you must have permissions to perform the required IAM actions. For more information, see Permissions required to access IAM resources in the IAM User Guide.

To get your access key ID and secret access key.

1. Open the IAM console.
2. On the navigation menu, choose **Users**.
3. Choose your IAM user name (not the check box).
4. Open the **Security credentials** tab, and then choose **Create access key**.
5. To see the new access key, choose **Show**. Your credentials resemble the following:
   - Access key ID: `AKIAIOSFODNN7EXAMPLE`
   - Secret access key: `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
6. To download the key pair, choose **Download .csv file**. Store the keys in a secure location.

> **Warning**
>
> Keep the keys confidential to protect your AWS account, and never share them with anyone outside your organization.

Related topics

- What Is IAM? in IAM User Guide.
- AWS Security Credentials in Amazon Web Services General Reference.

Invoke an Operation

After you have installed the SDK, you import AWS packages into your Go applications to use the SDK, as shown in the following example, which imports the AWS, Config, and Amazon S3 libraries. After importing the SDK packages, the AWS SDK Shared Configuration is loaded, a client is constructed, and an API operation is invoked.

```
package main

import (
    "context"
    "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/s3"
)

func main() {
    // Load the Shared AWS Configuration (~/.aws/config)
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatal(err)
    }

    // Create an Amazon S3 service client
    client := s3.NewFromConfig(cfg)

    // Get the first page of results for ListObjectsV2 for a bucket
    output, err := client.ListObjectsV2(context.TODO(), &s3.ListObjectsV2Input{
        Bucket: aws.String("amzn-s3-demo-bucket"),
    })
    if err != nil {
        log.Fatal(err)
    }

    log.Println("first page results")
```

```
    for _, object := range output.Contents {
        log.Printf("key=%s size=%d", aws.ToString(object.Key), *object.Size)
    }
}
```

Configure the SDK

In the AWS SDK for Go V2, you can configure common settings for service clients, such as the logger, log level, and retry configuration. Most settings are optional. However, for each service client, you must specify an AWS Region and your credentials. The SDK uses these values to send requests to the correct Region and sign requests with the correct credentials. You can specify these values as programmatically in code or via the execution environment.

Loading AWS Shared Configuration Files

There are a number of ways to initialize a service API client, but the following is the most common pattern recommended to users.

To configure the SDK to use the AWS shared configuration files, use the following code:

```
import (
  "context"
  "log"
  "github.com/aws/aws-sdk-go-v2/config"
)

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
  log.Fatalf("failed to load configuration, %v", err)
}
```

`config.LoadDefaultConfig(context.TODO())` will construct an aws.Config using the AWS shared configuration sources. This includes configuring a credential provider, configuring the AWS Region, and loading service specific configuration. Service clients can be constructed using the loaded `aws.Config`, providing a consistent pattern for constructing clients.

For more information about AWS shared configuration files, see Configuration in the AWS SDKs and Tools Reference Guide.

Specifying the AWS Region

When you specify the Region, you specify where to send requests, such as `us-west-2` or `us-east-2`. For a list of Regions for each service, see Service endpoints and quotas in the Amazon Web Services General Reference.

The SDK does not have a default Region. To specify a Region:

- Set the `AWS_REGION` environment variable to the default Region.
- Set the region explicitly using config.WithRegion as an argument to `config.LoadDefaultConfig` when loading configuration.

REVIEW: If you set a Region using all of these techniques, the SDK uses the Region you explicitly specified.

Configure Region with Environment Variable

Linux, macOS, or Unix

```
export AWS_REGION=us-west-2
```

Windows

```
set AWS_REGION=us-west-2
```

Specify Region Programmatically

```
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-west-2"))
```

Specifying Credentials

The AWS SDK for Go requires credentials (an access key and secret access key) to sign requests to AWS. You can specify your credentials in several locations, depending on your particular use case. For information about obtaining credentials, see Get started with AWS SDK for Go.

When you initialize an `aws.Config` instance using `config.LoadDefaultConfig`, the SDK uses its default credential chain to find AWS credentials. This default credential chain looks for credentials in the following order:

1. Environment variables.
    1. Static Credentials (`AWS_ACCESS_KEY_ID`, `AWS_SECRET_ACCESS_KEY`, `AWS_SESSION_TOKEN`)
    2. Web Identity Token (`AWS_WEB_IDENTITY_TOKEN_FILE`)
2. Shared configuration files.
    1. SDK defaults to `credentials` file under `.aws` folder that is placed in the home folder on your computer.
    2. SDK defaults to `config` file under `.aws` folder that is placed in the home folder on your computer.
3. If your application uses an Amazon ECS task definition or RunTask API operation, IAM role for tasks.
4. If your application is running on an Amazon EC2 instance, IAM role for Amazon EC2.

The SDK detects and uses the built-in providers automatically, without requiring manual configurations. For example, if you use IAM roles for Amazon EC2 instances, your applications automatically use the instance's credentials. You don't need to manually configure credentials in your application.

As a best practice, AWS recommends that you specify credentials in the following order:

1. Use IAM roles for tasks if your application uses an Amazon ECS task definition or RunTask API operation.
2. Use IAM roles for Amazon EC2 (if your application is running on an Amazon EC2 instance).

    IAM roles provide applications on the instance temporary security credentials to make AWS calls. IAM roles provide an easy way to distribute and manage credentials on multiple Amazon EC2 instances.
3. Use shared credentials or config files.

    The credentials and config files are shared across other AWS SDKs and AWS CLI. As a security best practice, we recommend using credentials file for setting sensitive values such as access key IDs and secret keys. Here are the formatting requirements for each of these files.
4. Use environment variables.

    Setting environment variables is useful if you're doing development work on a machine other than an Amazon EC2 instance.

IAM Roles for Tasks

If your application uses an Amazon ECS task definition or `RunTask` operation, use IAM Roles for Tasks to specify an IAM role that can be used by the containers in a task.

IAM Roles for Amazon EC2 Instances

If you are running your application on an Amazon EC2 instance, use the instance's IAM role to get temporary security credentials to make calls to AWS.

If you have configured your instance to use IAM roles, the SDK uses these credentials for your application automatically. You don't need to manually specify these credentials.

Shared Credentials and Configuration

The shared credentials and config files can be used to share common configuration amongst AWS SDKs and other tools. If you use different credentials for different tools or applications, you can use *profiles* to configure multiple access keys in the same configuration file.

You can provide multiple credential or config files locations using `config.LoadOptions`, by default the SDK loads files stored at default locations mentioned in the Specifying Credentials.

```
import (
    "context"
    "github.com/aws/aws-sdk-go-v2/config"
)

// ...

cfg , err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedCredentialsFiles(
    []string{"test/credentials", "data/credentials"},
    ),
    config.WithSharedConfigFiles(
        []string{"test/config", "data/config"},
    )
)
```

When working with shared credentials and config files, if duplicate profiles are specified they are merged to resolve a profile. In case of merge conflict,

1. If duplicate profiles are specified within a same credentials/config file, the profile properties specified in the latter profile takes precedence.
2. If duplicate profiles are specified across either multiple credentials files or across multiple config files, the profile properties are resolved as per the order of file input to the `config.LoadOptions`. The profile properties in the latter files take precedence.
3. If a profile exists in both credentials file and config file, the credentials file properties take precedence.

If needed, you can enable `LogConfigurationWarnings` on `config.LoadOptions` and log the profile resolution steps.

Creating the Credentials File

If you don't have a shared credentials file (`.aws/credentials`), you can use any text editor to create one in your home directory. Add the following content to your credentials file, replacing *<YOUR_ACCESS_KEY_ID>* and *<YOUR_SECRET_ACCESS_KEY>* with your credentials.

```
[default]
aws_access_key_id = <YOUR_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_SECRET_ACCESS_KEY>
```

The `[default]` heading defines credentials for the default profile, which the SDK will use unless you configure it to use another profile.

You can also use temporary security credentials by adding the session tokens to your profile, as shown in the following example:

```
[temp]
aws_access_key_id = <YOUR_TEMP_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_TEMP_SECRET_ACCESS_KEY>
aws_session_token = <YOUR_SESSION_TOKEN>
```

The section name for a non-default profile within a credentials file must not begin with the word `profile`. You can read more at AWS SDKs and Tools Reference Guide.

Creating the Config File

If you don't have a shared credentials file (`.aws/config`), you can use any text editor to create one in your home directory. Add the following content to your config file, replacing *<REGION>* with the desired region.

```
[default]
region = <REGION>
```

The `[default]` heading defines config for the default profile, which the SDK will use unless you configure it to use another profile.

You can use named profiles as shown in the following example:

```
[profile named-profile]
region = <REGION>
```

The section name for a non-default profile within a config file must always begin with the word `profile`, followed by the intended profile name. You can read more at the AWS SDKs and Tools Reference Guide.

Specifying Profiles

You can include multiple access keys in the same configuration file by associating each set of access keys with a profile. For example, in your credentials file, you can declare multiple profiles, as follows.

```
[default]
aws_access_key_id = <YOUR_DEFAULT_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_DEFAULT_SECRET_ACCESS_KEY>

[test-account]
aws_access_key_id = <YOUR_TEST_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_TEST_SECRET_ACCESS_KEY>

[prod-account]
; work profile
aws_access_key_id = <YOUR_PROD_ACCESS_KEY_ID>
aws_secret_access_key = <YOUR_PROD_SECRET_ACCESS_KEY>
```

By default, the SDK checks the `AWS_PROFILE` environment variable to determine which profile to use. If no `AWS_PROFILE` variable is set, the SDK uses the `default` profile.

Sometimes, you may want to use a different profile with your application. For example, you want to use the `test-account` credentials with your `myapp` application. You can use this profile by using the following command:

```
$ AWS_PROFILE=test-account myapp
```

You can also use instruct the SDK to select a profile by either calling `os.Setenv("AWS_PROFILE", "test-account")` before calling `config.LoadDefaultConfig`, or by passing an explicit profile as an argument as shown in the following example:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedConfigProfile("test-account"))
```

> **Note**
>
> If you specify credentials in environment variables, the SDK always uses those credentials, no matter which profile you specify.

Environment Variables

By default, the SDK detects AWS credentials set in your environment and uses them to sign requests to AWS. That way you don't need to manage credentials in your applications.

The SDK looks for credentials in the following environment variables:

- `AWS_ACCESS_KEY_ID`
- `AWS_SECRET_ACCESS_KEY`
- `AWS_SESSION_TOKEN` (optional)

The following examples show how you configure the environment variables.

Linux, OS X, or Unix

```
$ export AWS_ACCESS_KEY_ID=YOUR_AKID
```

```
$ export AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
$ export AWS_SESSION_TOKEN=TOKEN
```

Windows

```
> set AWS_ACCESS_KEY_ID=YOUR_AKID
> set AWS_SECRET_ACCESS_KEY=YOUR_SECRET_KEY
> set AWS_SESSION_TOKEN=TOKEN
```

Specify Credentials Programmatically

`config.LoadDefaultConfig` allows you to provide an explicit aws.CredentialProvider when loading the shared configuration sources. To pass an explicit credential provider when loading shared configuration use config.WithCredentialsProvider. For example, if `customProvider` references an instance of `aws.CredentialProvider` implementation, it can be passed during configuration loading like so:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithCredentialsProvider(customProvider))
```

If you explicitly provide credentials, as in this example, the SDK uses only those credentials.

> **Note**
>
> All credential providers passed to or returned by `LoadDefaultConfig` are wrapped in a CredentialsCache automatically. This enables caching and credential rotation that is concurrency safe. If you explicitly configure a provider on `aws.Config` directly, you must also explicitly wrap the provider with this type using NewCredentialsCache.

Static Credentials

You can hard-code credentials in your application by using the credentials.NewStaticCredentialsProvider credential provider to explicitly set the access keys to be used. For example:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider("AKID",
 "SECRET_KEY", "TOKEN")),
```

```
)
```

> **Warning**
>
> Do not embed credentials inside an application. Use this method only for testing purposes.

Single Sign-on Credentials

The SDK provides a credential provider for retrieving temporary AWS credentials using AWS IAM Identity Center. Using the AWS CLI, you authenticate with the AWS access portal and authorize access to temporary AWS credentials. You then configure your application to load the single sign-on (SSO) profile, and the SDK uses your SSO credentials to retrieve temporary AWS credentials that will be automatically renewed if expired. If your SSO credentials expire, you must explicitly renew them by logging in to your IAM Identity Center account again using the AWS CLI.

For example, you can create a profile, `dev-profile`, authenticate and authorize that profile using the AWS CLI, and configure your application as shown below.

1. First create the `profile` and `sso-session`

```
[profile dev-profile]
sso_session = dev-session
sso_account_id = 012345678901
sso_role_name = Developer
region = us-east-1

[sso-session dev-session]
sso_region = us-west-2
sso_start_url = https://company-sso-portal.awsapps.com/start
sso_registration_scopes = sso:account:access
```

2. Login using the AWS CLI to authenticate and authorize the SSO profile.

```
$ aws --profile dev-profile sso login
Attempting to automatically open the SSO authorization page in your default browser.
If the browser does not open or you wish to use a different device to authorize this
 request, open the following URL:
```

```
https://device.sso.us-west-2.amazonaws.com/

Then enter the code:

ABCD-EFGH
Successully logged into Start URL: https://company-sso-portal.awsapps.com/start
```

3. Next configure your application to use the SSO profile.

```
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(
    context.Background(),
    config.WithSharedConfigProfile("dev-profile"),
)
if err != nil {
    return err
}
```

For more information on configuring SSO profiles and authenticating using the AWS CLI see Configuring the AWS CLI to use AWS IAM Identity Center in the AWS CLI User Guide. For more information on programmatically constructing the SSO credential provider see the ssocreds API reference documentation.

Login credentials

You can use your existing AWS Management Console sign-in credentials for programmatic access to AWS services. After a browser-based authentication flow, AWS generates temporary credentials that work across local development tools like the AWS CLI, AWS Tools for PowerShell and AWS SDKs. This feature simplifies the process of configuring and managing AWS CLI credentials, especially if you prefer interactive authentication over managing long-term access keys.

1. Initiate the login flow using the AWS CLI, and follow the browser prompts. In this example, we save the login session to a new profile **dev-profile**, but this is optional.

   ```
   $ aws --profile dev-profile login
   ```

2. (Optional) Inspect your AWS shared configuration file to verify that the session was established.

```
[profile dev-profile]
login_session = arn:aws:sts::account id>:role
```

3. Next configure your application to use the login profile.

```
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(
    context.Background(),
    // only necessary if login session is saved to a non-default profile
    config.WithSharedConfigProfile("dev-profile"),
)
if err != nil {
    return err
}
```

For more information on configuring login profiles and authenticating using the AWS CLI, see Login for AWS local development using console credentials.

Other Credentials Providers

The SDK provides other methods for retrieving credentials in the credentials module. For example, you can retrieve temporary security credentials from AWS Security Token Service or credentials from encrypted storage.

Available Credential Providers:

- ec2rolecreds – Retrieve Credentials from Amazon EC2 Instances Roles via Amazon EC2 IMDS.
- endpointcreds – Retrieve Credentials from an arbitrary HTTP endpoint.
- processcreds – Retrieve Credentials from an external process that will be invoked by the host environment's shell.
- stscreds – Retrieve Credentials from AWS STS

Configure Authentication

The AWS SDK for Go provides the ability to configure the authentication behavior service. In most cases, the default configuration will suffice, but configuring custom authentication allows for additional behavior such as working with pre-release service features.

Definitions

This section provides a high-level description of authentication components in the AWS SDK for Go.

AuthScheme

An AuthScheme is the interface that defines the workflow through which the SDK retrieves a caller identity and attaches it to an operation request.

An auth scheme uses the following components, described in detail further below:

- A unique ID which identifies the scheme
- An identity resolver, which returns a caller identity used in the signing process (e.g. your AWS credentials)
- A signer, which performs the actual injection of caller identity into the operation's transport request (e.g. the `Authorization` HTTP header)

Each service client options includes an `AuthSchemes` field, which by default is populated with the list of auth schemes supported by that service.

AuthSchemeResolver

Each service client options includes an `AuthSchemeResolver` field. This interface, defined per-service, is the API called by the SDK to determine the possible authentication options for each operation.

> **Important**
>
> The auth scheme resolver does NOT dictate what auth scheme is used. It returns a list of schemes that *can* be used ("options"), the final scheme is selected through a fixed algorithm described here.

Option

Returned from a call to `ResolverAuthSchemes`, an Option represents a possible authentication option.

An option consists of three sets of information:

- An ID representing the possible scheme
- An opaque set of properties to be provided to the scheme's identity resolver
- An opaque set of properties to be provided to the scheme's signer

A note on properties

For 99% of use cases, callers need not be concerned with the opaque properties for identity resolution and signing. The SDK will pull out the necessary properties for each scheme and pass them to the strongly-typed interfaces exposed in the SDK. For example, the default auth resolver for services encode the SigV4 option to have signer properties for the signing name and region, the values of which are passed to the client's configured v4.HTTPSigner implementation when SigV4 is selected.

Identity

An Identity is an abstract representation of who the SDK caller is.

The most common type of identity used in the SDK is a set of `aws.Credentials`. For most use cases, the caller need not concern themselves with `Identity` as an abstraction and can work with the concrete types directly.

> **Note**
>
> To preserve backwards compatibility and prevent API confusion, the AWS SDK-specific identity type `aws.Credentials` does not directly satisfy the `Identity` interface. This mapping is handled internally.

IdentityResolver

IdentityResolver is the interface through which an `Identity` is retrieved.

Concrete versions of `IdentityResolver` exist in the SDK in strongly-typed form (e.g. aws.CredentialsProvider), the SDK handles this mapping internally.

A caller will only need to directly implement the `IdentityResolver` interface when defining an external auth scheme.

Signer

Signer is the interface through which a request is supplemented with the retrieved caller `Identity`.

Concrete versions of `Signer` exist in the SDK in strongly-typed form (e.g. v4.HTTPSigner), the SDK handles this mapping internally.

A caller will only need to directly implement the `Signer` interface when defining an external auth scheme.

AuthResolverParameters

Each service takes a specific set of inputs which are passed to its resolution function, defined in each service package as `AuthResolverParameters`.

The base resolver parameters are as follows:

name	type	description
`Operation`	`string`	The name of the operation being invoked.
`Region`	`string`	The client's AWS region. Only present for services that use SigV4[A].

If you are implementing your own resolver, you should never need to construct your own instance of its parameters. The SDK will source these values per-request and pass them to your implementation.

Auth scheme resolution workflow

When you call an AWS service operation through the SDK, the following sequence of actions occurs after the request has been serialized:

1. The SDK calls the client's `AuthSchemeResolver.ResolveAuthSchemes()` API, sourcing the input parameters as necessary, to obtain a list of possible Options for the operation.
2. The SDK iterates over that list and selects the first scheme that satisfies the following conditions.
   - A scheme with matching ID is present in the client's own `AuthSchemes` list
   - The scheme's identity resolver exists (is non-`nil`) on the client's Options (checked via the scheme's `GetIdentityResolver` method, the mapping to the concrete identity resolver types described above is handled internally) (1)
3. Assuming a viable scheme was selected, the SDK invokes its `GetIdentityResolver()` API to retrieve the caller's identity. For example, the builtin SigV4 auth scheme will map to the client's `Credentials` provider internally.
4. The SDK calls the identity resolver's `GetIdentity()` (e.g. `aws.CredentialProvider.Retrieve()` for SigV4).
5. The SDK calls the endpoint resolver's `ResolveEndpoint()` to find the endpoint for the request. The endpoint may include additional metadata that influences the signing process (e.g. unique signing name for S3 Object Lambda).
6. The SDK calls the auth scheme's `Signer()` API to retrieve its signer, and uses its `SignRequest()` API to sign the request with the previously-retrieved caller identity.

(1) If the SDK encounters the anonymous option (ID `smithy.api#noAuth`) in the list, it is selected automatically, as there is no corresponding identity resolver.

Natively-supported `AuthSchemes`

The following auth schemes are natively supported by AWS SDK for Go.

Name	Scheme ID	Identity resolver	Signer	Notes
SigV4	`aws.auth#sigv4`	aws.CredentialsProvider	v4.HTTPSigner	The current default for most

Name	Scheme ID	Identity resolver	Signer	Notes
				AWS service operations.
SigV4A	`aws.auth#sigv4a`	aws.CredentialsProvider	n/a	SigV4A usage is limited at this time, the signer implementation is internal. See this announcement for a new opt-in module *aws-http-auth* that exposes general purpose APIs for signing HTTP requests.
SigV4Express	`com.amazonaws.s3#sigv4express`	s3.ExpressCredentialsProvider	v4.HTTPSigner	Used for Express One Zone.
HTTP Bearer	`smithy.api#httpBearerAuth`	smithybearer.TokenProvider	smithybearer.Signer	Used by codecatalyst.
Anonymous	`smithy.api#noAuth`	n/a	n/a	No authentication - no identity is required, and the request is not signed or authenticated.

Identity configuration

In AWS SDK for Go, the identity components of an auth scheme are configured in SDK client `Options`. The SDK will automatically pick up and use the values for these components for the scheme it selects when an operation is called.

> **Note**
>
> For backwards compatibility reasons, the SDK implicitly allows the use of the anonymous auth scheme if no identity resolvers are configured. This can be manually achieved by setting all identity resolvers on a client's `Options` to `nil` (the sigv4 identity resolver can also be set to `aws.AnonymousCredentials{}`).

Signer configuration

In AWS SDK for Go, the signer components of an auth scheme are configured in SDK client `Options`. The SDK will automatically pick up and use the values for these components for the scheme it selects when an operation is called. No additional configuration is necessary.

Custom auth scheme

In order to define a custom auth scheme and configure it for use, the caller must do the following:

1. Define an AuthScheme implementation
2. Register the scheme on the SDK client's `AuthSchemes` list
3. Instrument the SDK client's `AuthSchemeResolver` to return an auth `Option` with the scheme's ID where applicable

> **Warning**
>
> The following services have unique or customized authentication behavior. We recommend you delegate to the default implementation and wrap accordingly if you require custom authentication behavior:

Service	Notes
S3	Conditional use of SigV4A and SigV4Express depending on operation input.
EventBridge	Conditional use of SigV4A depending on operation input.
Cognito	Certain operations are anonymous-only.
SSO	Certain operations are anonymous-only.
STS	Certain operations are anonymous-only.

Configure Client Endpoints

Warning

Endpoint resolution is an advanced SDK topic. By changing these settings you risk breaking your code. The default settings should be applicable to most users in production environments.

The AWS SDK for Go provides the ability to configure a custom endpoint to be used for a service. In most cases, the default configuration will suffice. Configuring custom endpoints allows for additional behavior, such as working with pre-release versions of a service.

Customization

There are two "versions" of endpoint resolution config within the SDK.

- v2, released in Q3 of 2023, configured via:
  - `EndpointResolverV2`
  - `BaseEndpoint`
- v1, released alongside the SDK, configured via:
  - `EndpointResolver`

We recommend users of v1 endpoint resolution migrate to v2 to obtain access to newer endpoint-related service features.

V2: `EndpointResolverV2` + `BaseEndpoint`

In resolution v2, `EndpointResolverV2` is the definitive mechanism through which endpoint resolution occurs. The resolver's `ResolveEndpoint` method is invoked as part of the workflow for every request you make in the SDK. The hostname of the `Endpoint` returned by the resolver is used **as-is** when making the request (operation serializers can still append to the HTTP path, however).

Resolution v2 includes an additional client-level config, `BaseEndpoint`, which is used to specify a "base" hostname for the instance of your service. The value set here is not definitive-- it is ultimately passed as a parameter to the client's `EndpointResolverV2` when final resolution occurs (read on for more information about `EndpointResolverV2` parameters). The resolver implementation then has the opportunity to inspect and potentially modify that value to determine the final endpoint.

For example, if you perform an S3 `GetObject` request against a given bucket with a client where you've specified a `BaseEndpoint`, the default resolver will inject the bucket into the hostname if it is virtual-host compatible (assuming you haven't disabled virtual-hosting in client config).

In practice, `BaseEndpoint` will most likely be used to point your client at a development or preview instance of a service.

`EndpointResolverV2` parameters

Each service takes a specific set of inputs which are passed to its resolution function, defined in each service package as `EndpointParameters`.

Every service includes the following base parameters, which are used to facilitate general endpoint resolution within AWS:

name	type	description
`Region`	`string`	The client's AWS region
`Endpoint`	`string`	The value set for `BaseEndpoint` in client config

name	type	description
`UseFips`	`bool`	Whether FIPS endpoints are enabled in client config
`UseDualStack`	`bool`	Whether dual-stack endpoints are enabled in client config

Services can specify additional parameters required for resolution. For example, S3's `EndpointParameters` include the bucket name, as well as several S3-specific feature settings such as whether virtual host addressing is enabled.

If you are implementing your own `EndpointResolverV2`, you should never need to construct your own instance of `EndpointParameters`. The SDK will source the values per-request and pass them to your implementation.

A note about Amazon S3

Amazon S3 is a complex service with many of its features modeled through complex endpoint customizations, such as bucket virtual hosting, S3 MRAP, and more.

Because of this, we recommend that you don't replace the `EndpointResolverV2` implementation in your S3 client. If you need to extend its resolution behavior, perhaps by sending requests to a local development stack with additional endpoint considerations, we recommend wrapping the default implementation such that it delegates back to the default as a fallback (shown in examples below).

Examples

With `BaseEndpoint`

The following code snippet shows how to point your S3 client at a local instance of a service, which in this example is hosted on the loopback device at port 8080.

```
client := s3.NewFromConfig(cfg, func (o *svc.Options) {
    o.BaseEndpoint = aws.String("https://localhost:8080/")
})
```

With `EndpointResolverV2`

The following code snippet shows how to inject custom behavior into S3's endpoint resolution using `EndpointResolverV2`.

```
import (
    "context"
    "net/url"

    "github.com/aws/aws-sdk-go-v2/service/s3"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
)

type resolverV2 struct {
    // you could inject additional application context here as well
}

func (*resolverV2) ResolveEndpoint(ctx context.Context, params s3.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    if /* input params or caller context indicate we must route somewhere */ {
        u, err := url.Parse("https://custom.service.endpoint/")
        if err != nil {
            return smithyendpoints.Endpoint{}, err
        }
        return smithyendpoints.Endpoint{
            URI: *u,
        }, nil
    }

    // delegate back to the default v2 resolver otherwise
    return s3.NewDefaultEndpointResolverV2().ResolveEndpoint(ctx, params)
}

func main() {
    // load config...

    client := s3.NewFromConfig(cfg, func (o *s3.Options) {
        o.EndpointResolverV2 = &resolverV2{
            // ...
        }
    })
}
```

With both

The following sample program demonstrates the interaction between `BaseEndpoint` and `EndpointResolverV2`. **This is an advanced use case:**

```
import (
    "context"
    "fmt"
    "log"
    "net/url"

    "github.com/aws/aws-sdk-go-v2"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
)

type resolverV2 struct {}

func (*resolverV2) ResolveEndpoint(ctx context.Context, params s3.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    // s3.Options.BaseEndpoint is accessible here:
    fmt.Printf("The endpoint provided in config is %s\n", *params.Endpoint)

    // fallback to default
    return s3.NewDefaultEndpointResolverV2().ResolveEndpoint(ctx, params)
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if (err != nil) {
        log.Fatal(err)
    }

    client := s3.NewFromConfig(cfg, func (o *s3.Options) {
        o.BaseEndpoint = aws.String("https://endpoint.dev/")
        o.EndpointResolverV2 = &resolverV2{}
    })

    // ignore the output, this is just for demonstration
    client.ListBuckets(context.Background(), nil)
}
```

When run, the above program outputs the following:

```
The endpoint provided in config is https://endpoint.dev/
```

V1: EndpointResolver

> **Warning**
>
> Endpoint resolution v1 is retained for backwards compatibility and is isolated from the modern behavior in endpoint resolution v2. It will only be used if the `EndpointResolver` field is set by the caller.
> Use of v1 will most likely prevent you from accessing endpoint-related service features introduced with or after the release of v2 resolution. See "Migration" for instructions on how to upgrade.

A EndpointResolver can be configured to provide custom endpoint resolution logic for service clients. You can use a custom endpoint resolver to override a service's endpoint resolution logic for all endpoints, or a just specific regional endpoint. Custom endpoint resolver can trigger the service's endpoint resolution logic to fallback if a custom resolver does not wish to resolve a requested endpoint. EndpointResolverWithOptionsFunc can be used to easily wrap functions to satisfy the `EndpointResolverWithOptions` interface.

A `EndpointResolver` can be easily configured by passing the resolver wrapped with WithEndpointResolverWithOptions to LoadDefaultConfig , allowing for the ability to override endpoints when loading credentials, as well as configuring the resulting `aws.Config` with your custom endpoint resolver.

The endpoint resolver is given the service and region as a string, allowing for the resolver to dynamically drive its behavior. Each service client package has an exported `ServiceID` constant which can be used to determine which service client is invoking your endpoint resolver.

An endpoint resolver can use the EndpointNotFoundError sentinel error value to trigger fallback resolution to the service clients default resolution logic. This allows you to selectively override one or more endpoints seamlessly without having to handle fallback logic.

If your endpoint resolver implementation returns an error other than `EndpointNotFoundError`, endpoint resolution will stop and the service operation returns an error to your application.

Examples

With fallback

The following code snippet shows how a single service endpoint can be overridden for DynamoDB with fallback behavior for other endpoints:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
        }, nil
    }
    // returning EndpointNotFoundError will allow the service to fallback to it's
 default resolution
    return aws.Endpoint{}, &aws.EndpointNotFoundError{}
})

cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Without fallback

The following code snippet shows how a single service endpoint can be overridden for DynamoDB without fallback behavior for other endpoints:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
        }, nil
    }
    return aws.Endpoint{}, fmt.Errorf("unknown endpoint requested")
})
```

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Immutable endpoints

> **Warning**
>
> Setting an endpoint as immutable may prevent some service client features from functioning correctly, and could result in undefined behavior. Caution should be taken when defining an endpoint as immutable.

Some service clients, such as Amazon S3, may modify the endpoint returned by the resolver for certain service operations. For example, Amazon S3 will automatically handle Virtual Bucket Addressing by mutating the resolved endpoint. You can prevent the SDK from mutating your custom endpoints by setting HostnameImmutable to `true`. For example:

```
customResolver := aws.EndpointResolverWithOptionsFunc(func(service, region string,
 options ...interface{}) (aws.Endpoint, error) {
    if service == dynamodb.ServiceID && region == "us-west-2" {
        return aws.Endpoint{
            PartitionID:   "aws",
            URL:           "https://test.us-west-2.amazonaws.com",
            SigningRegion: "us-west-2",
            HostnameImmutable: true,
        }, nil
    }
    return aws.Endpoint{}, fmt.Errorf("unknown endpoint requested")
})

cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithEndpointResolverWithOptions(customResolver))
```

Migration

When migrating from v1 to v2 of endpoint resolution, the following general principles apply:

- Returning an Endpoint with HostnameImmutable set to `false` is roughly equivalent to setting `BaseEndpoint` to the originally returned URL from v1 and leaving `EndpointResolverV2` as the default.

- Returning an Endpoint with HostnameImmutable set to `true` is roughly equivalent to implementing an `EndpointResolverV2` which returns the originally returned URL from v1.
  - The primary exception is for operations with modeled endpoint prefixes. A note on this is given further down.

Examples for these cases are provided below.

> **Warning**
>
> V1 immutable endpoints and V2 resolution are not equivalent in behavior. For example, signing overrides for custom features like S3 Object Lambda would still be set for immutable endpoints returned via v1 code, but the same will not be done for v2.

Note on host prefixes

Some operations are modeled with host prefixes to be prepended to the resolved endpoint. This behavior must work in tandem with the output of ResolveEndpointV2 and therefore the host prefix will still be applied to that result.

You can manually disable endpoint host prefixing by applying a middleware, see the examples section.

Examples

Mutable endpoint

The following code sample demonstrates how to migrate a basic v1 endpoint resolver that returns a modifiable endpoint:

```
// v1
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolver = svc.EndpointResolverFromURL("https://custom.endpoint.api/")
})

// v2
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    // the value of BaseEndpoint is passed to the default EndpointResolverV2
    // implementation, which will handle routing for features such as S3 accelerate,
    // MRAP, etc.
    o.BaseEndpoint = aws.String("https://custom.endpoint.api/")
```

```
})
```

Immutable endpoint

```
// v1
client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolver = svc.EndpointResolverFromURL("https://custom.endpoint.api/",
 func (e *aws.Endpoint) {
        e.HostnameImmutable = true
    })
})

// v2
import (
    smithyendpoints "github.com/aws/smithy-go/endpoints"
)

type staticResolver struct {}

func (*staticResolver) ResolveEndpoint(ctx context.Context, params
 svc.EndpointParameters) (
        smithyendpoints.Endpoint, error,
    ) {
    // This value will be used as-is when making the request.
    u, err := url.Parse("https://custom.endpoint.api/")
    if err != nil {
        return smithyendpoints.Endpoint{}, err
    }
    return smithyendpoints.Endpoint{
        URI: *u,
    }, nil
}

client := svc.NewFromConfig(cfg, func (o *svc.Options) {
    o.EndpointResolverV2 = &staticResolver{}
})
```

Disable host prefix

```
import (
    "context"
    "fmt"
    "net/url"
```

```
    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/<service>"
    smithyendpoints "github.com/aws/smithy-go/endpoints"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

// disableEndpointPrefix applies the flag that will prevent any
// operation-specific host prefix from being applied
type disableEndpointPrefix struct{}

func (disableEndpointPrefix) ID() string { return "disableEndpointPrefix" }

func (disableEndpointPrefix) HandleInitialize(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (middleware.InitializeOutput, middleware.Metadata, error) {
    ctx = smithyhttp.SetHostnameImmutable(ctx, true)
    return next.HandleInitialize(ctx, in)
}

func addDisableEndpointPrefix(o *<service>.Options) {
    o.APIOptions = append(o.APIOptions, (func(stack *middleware.Stack) error {
        return stack.Initialize.Add(disableEndpointPrefix{}, middleware.After)
    }))
}

type staticResolver struct{}

func (staticResolver) ResolveEndpoint(ctx context.Context, params
 <service>.EndpointParameters) (
    smithyendpoints.Endpoint, error,
) {
    u, err := url.Parse("https://custom.endpoint.api/")
    if err != nil {
        return smithyendpoints.Endpoint{}, err
    }

    return smithyendpoints.Endpoint{URI: *u}, nil
}
```

```
func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        panic(err)
    }

    svc := <service>.NewFromConfig(cfg, func(o *<service>.Options) {
        o.EndpointResolverV2 = staticResolver{}
    })

    _, err = svc.<Operation>(context.Background(), &<service>.<OperationInput>{ /* ...
 */ },
        addDisableEndpointPrefix)
    if err != nil {
        panic(err)
    }
}
```

Customize the HTTP Client

The AWS SDK for Go uses a default HTTP client with default configuration values. Although you can change some of these configuration values, the default HTTP client and transport are not sufficiently configured for customers using the AWS SDK for Go in an environment with high throughput and low latency requirements. For more information, please refer to the Frequently Asked Questions as configuration recommendations vary based on specific workloads. This section describes how to configure a custom HTTP client, and use that client to create AWS SDK for Go calls.

To assist you in creating a custom HTTP client, this section describes how to the NewBuildableClient to configure custom settings, and use that client with an AWS SDK for Go service client.

Let's define what we want to customize.

Overriding During Configuration Loading

Custom HTTP clients can be provided when calling LoadDefaultConfig by wrapping the client using WithHTTPClient and passing the resulting value to `LoadDefaultConfig`. For example, to pass `customClient` as our client:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithHTTPClient(customClient))
```

Timeout

The `BuildableHTTPClient` can be configured with a request timeout limit. This timeout includes the time to connect, process any redirects, and read the complete response body. For example, to modify the client timeout:

```
import "github.com/aws/aws-sdk-go-v2/aws/transport/http"

// ...

httpClient := http.NewBuildableClient().WithTimeout(time.Second*5)
```

Dialer

The `BuildableHTTPClient` provides a builder mechanics for constructing clients with modified Dialer options. The following example shows how to configure a clients `Dialer` settings.

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net"

// ...

httpClient := awshttp.NewBuildableClient().WithDialerOptions(func(d *net.Dialer) {
    d.KeepAlive = -1
    d.Timeout = time.Millisecond*500
})
```

Settings

Dialer.KeepAlive

This setting represents the keep-alive period for an active network connection.

Set to a negative value to disable keep-alives.

Set to **0** to enable keep-alives if supported by the protocol and operating system.

Network protocols or operating systems that do not support keep-alives ignore this field. By default, TCP enables keep alive.

See https://golang.org/pkg/net/#Dialer.KeepAlive

Set `KeepAlive` as **time.Duration**.

Dialer.Timeout

This setting represents the maximum amount of time a dial waits for a connection to be created.

Default is 30 seconds.

See https://golang.org/pkg/net/#Dialer.Timeout

Set `Timeout` as **time.Duration**.

Transport

The `BuildableHTTPClient` provides a builder mechanics for constructing clients with modified Transport options.

Configuring a Proxy

If you cannot directly connect to the internet, you can use Go-supported environment variables (`HTTP_PROXY`/`HTTPS_PROXY`) or create a custom HTTP client to configure your proxy. The following example configures the client to use `PROXY_URL` as the proxy endpoint:

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net/http"

// ...

httpClient := awshttp.NewBuildableClient().WithTransportOptions(func(tr
 *http.Transport) {
    proxyURL, err := url.Parse("PROXY_URL")
    if err != nil {
        log.Fatal(err)
    }
    tr.Proxy = http.ProxyURL(proxyURL)
})
```

Other Settings

Below are a few other `Transport` settings that can be modified to tune the HTTP client. Any additional settings not described here can be found in the Transport type documentation. These settings can be applied as shown in the following example:

```
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
import "net/http"

// ...

httpClient := awshttp.NewBuildableClient().WithTransportOptions(func(tr
 *http.Transport) {
    tr.ExpectContinueTimeout = 0
    tr.MaxIdleConns = 10
})
```

Transport.ExpectContinueTimeout

If the request has an "Expect: 100-continue" header, this setting represents the maximum amount of time to wait for a server's first response headers after fully writing the request headers, This time does not include the time to send the request header. The HTTP client sends its payload after this timeout is exhausted.

Default 1 second.

Set to **0** for no timeout and send request payload without waiting. One use case is when you run into issues with proxies or third party services that take a session similar to the use of Amazon S3 in the function shown later.

See https://golang.org/pkg/net/http/#Transport.ExpectContinueTimeout

Set `ExpectContinue` as **time.Duration**.

Transport.IdleConnTimeout

This setting represents the maximum amount of time to keep an idle network connection alive between HTTP requests.

Set to **0** for no limit.

See https://golang.org/pkg/net/http/#Transport.IdleConnTimeout

Set `IdleConnTimeout` as **time.Duration**.

Transport.MaxIdleConns

This setting represents the maximum number of idle (keep-alive) connections across all hosts. One use case for increasing this value is when you are seeing many connections in a short period from the same clients

0 means no limit.

See https://golang.org/pkg/net/http/#Transport.MaxIdleConns

Set`MaxIdleConns` as **int**.

Transport.MaxIdleConnsPerHost

This setting represents the maximum number of idle (keep-alive) connections to keep per-host. One use case for increasing this value is when you are seeing many connections in a short period from the same clients

Default is two idle connections per host.

Set to **0** to use DefaultMaxIdleConnsPerHost (2).

See https://golang.org/pkg/net/http/#Transport.MaxIdleConnsPerHost

Set `MaxIdleConnsPerHost` as **int**.

Transport.ResponseHeaderTimeout

This setting represents the maximum amount of time to wait for a client to read the response header.

If the client isn't able to read the response's header within this duration, the request fails with a timeout error.

Be careful setting this value when using long-running Lambda functions, as the operation does not return any response headers until the Lambda function has finished or timed out. However, you can still use this option with the ** InvokeAsync** API operation.

Default is no timeout; wait forever.

See https://golang.org/pkg/net/http/#Transport.ResponseHeaderTimeout

Set `ResponseHeaderTimeout` as **time.Duration**.

Transport.TLSHandshakeTimeout

This setting represents the maximum amount of time waiting for a TLS handshake to be completed.

Default is 10 seconds.

Zero means no timeout.

See https://golang.org/pkg/net/http/#Transport.TLSHandshakeTimeout

Set `TLSHandshakeTimeout` as **time.Duration**.

HTTP Interceptors

You can use interceptors to hook into the execution of API requests and responses. Interceptors are open-ended mechanisms in which the SDK calls code that you write to inject behavior into the request/response lifecycle. This way, you can modify an in-flight request, debug request processing, view exceptions, and more.

Interceptors vs. middleware

The AWS SDK for Go v2 provides both interceptors and middleware for customizing request processing. While both serve similar purposes, they are designed for different audiences and use cases:

- **Interceptors** are designed for SDK users who want to customize request/response processing with a simple, HTTP-focused API. They provide specific hook points in the request lifecycle and work directly with HTTP requests and responses.
- **Middleware** is a more advanced, transport-agnostic system primarily used internally by the SDK. While powerful, middleware requires deeper knowledge of SDK internals and involves more complex interfaces.

Key advantages of interceptors over middleware for common use cases:

- **HTTP-focused**: Interceptors work directly with HTTP requests and responses, eliminating the need for transport type checking that middleware requires.

- **Simpler interfaces**: Each interceptor hook has a specific, focused interface rather than the generic middleware pattern.
- **Clearer execution model**: Interceptors execute at well-defined points in the request lifecycle without requiring knowledge of middleware stack ordering.

> **Note**
>
> Interceptors are built on top of the existing middleware system, so both can coexist in the same application. Middleware remains available for advanced use cases that require transport-agnostic behavior or complex stack manipulation.

Available interceptor hooks

The AWS SDK for Go v2 provides interceptor hooks at various stages of the request lifecycle. Each hook corresponds to a specific interface that you can implement:

- `BeforeExecution` - First hook called during operation execution
- `BeforeSerialization` - Before input message is serialized into transport request
- `AfterSerialization` - After input message is serialized into transport request
- `BeforeRetryLoop` - Before entering the retry loop
- `BeforeAttempt` - First hook called inside retry loop
- `BeforeSigning` - Before transport request is signed
- `AfterSigning` - After transport request is signed
- `BeforeTransmit` - Before transport request is sent
- `AfterTransmit` - After receiving transport response
- `BeforeDeserialization` - Before transport response is deserialized
- `AfterDeserialization` - After unmarshalling transport response
- `AfterAttempt` - Last hook called inside retry loop
- `AfterExecution` - Last hook called during operation execution

You can implement multiple interfaces in a single interceptor to hook into multiple stages of the request lifecycle.

Interceptor registration

You register interceptors when you construct a service client or when you override configuration for a specific operation. The registration differs depending on whether you want the interceptor to apply to all operations for your client or only specific ones.

Interceptors are managed through an interceptor registry that provides methods to add and remove interceptors. The following example shows a simple interceptor that adds an AWS X-Ray trace ID header to outgoing requests before the signing process:

```
type recursionDetection struct{}

func (recursionDetection) BeforeSigning(ctx context.Context, in
 *smithyhttp.InterceptorContext) error {
    if traceID := os.Getenv("_X_AMZN_TRACE_ID"); traceID != "" {
        in.Request.Header.Set("X-Amzn-Trace-Id", traceID)
    }
    return nil
}

// use it on the client
svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.Interceptors.AddBeforeSigning(recursionDetection{})
})
```

The interceptor registry is added to client Options, which enables per-operation interceptor configuration:

```
// ... or use it per-operation
s3.ListBuckets(context.Background(), &s3.ListBucketsInput{
}, func(o *s3.Options) {
   o.Interceptors.AddBeforeSigning(recursionDetection{})
})
```

Global interceptor configuration

You can also register interceptors globally using the `config.LoadDefaultConfig` function with the appropriate `With*` options for each interceptor type. This applies the interceptor to all AWS service clients created from that configuration:

```
type myExecutionInterceptor struct{}
```

```
func (*myExecutionInterceptor) AfterExecution(ctx context.Context, in
 *smithyhttp.InterceptorContext) error {
    // Add your custom logic here
    return nil
}

cfg, err := config.LoadDefaultConfig(context.Background(),
    config.WithAfterExecution(&myExecutionInterceptor{}))
if err != nil {
    panic(err)
}

// every service client created from the above config
// will include this interceptor
svc := s3.NewFromConfig(cfg)
```

Logging

The AWS SDK for Go has logging facilities available that allow your application to enable debugging information for debugging and diagnosing request issues or failures. The Logger interface and ClientLogMode are the main components available to you for determining how and what should be logged by clients.

Logger

When constructing an Config using LoadDefaultConfig a default `Logger` is configured to send log messages to the process' standard error (stderr). A custom logger that satisfies the Logger interface can be passed as an argument to `LoadDefaultConfig` by wrapping it with config.WithLogger.

For example, to configure our clients to use our `applicationLogger`:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithLogger(applicationLogger))
```

Now clients configured using the constructed `aws.Config` will send log messages to `applicationLogger`.

Context-Aware Loggers

A Logger implementation may implement the optional ContextLogger interface. Loggers that implement this interface will have their `WithContext` methods invoked with the current context. This allows your logging implementations to return a new `Logger` that can write additional logging metadata based on values present in the context.

ClientLogMode

By default, service clients do not produce log messages. To configure clients to send log messages for debugging purposes, use the ClientLogMode member on `Config`. `ClientLogMode` can be set to enable debugging messaging for:

- Signature Version 4 (SigV4) Signing
- Request Retries
- HTTP Requests
- HTTP Responses

For example, to enable logging of HTTP requests and retries:

```
cfg, err := config.LoadDefaultConfig(context.TODO(),
 config.WithClientLogMode(aws.LogRetries | aws.LogRequest))
```

See ClientLogMode for the different client log modes available.

Traces and metrics

The AWS SDK for Go is instrumented to record trace spans and client-side metrics for SDK operations. By default, clients use no-op implementations for both tracing and metrics, meaning no data is collected unless you configure a provider.

Service clients have two configuration options for observability:

- TracerProvider – Entry point for creating tracers and recording client trace spans.
- MeterProvider – Entry point for creating meters and recording client-side metrics.

While inspired by the OpenTelemetry (OTel) specifications, these APIs are defined independently in `smithy-go`. The SDK provides adapter modules to connect concrete OTel SDK implementations to the SDK's provider interfaces.

Tracing

Use the smithyoteltracing adapter module to connect an OTel `trace.TracerProvider` to a service client.

The following example shows how to configure tracing for an Amazon S3 client:

```
import (
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/tracing/smithyoteltracing"
    "go.opentelemetry.io/otel/trace"
)

// provider is an OTel trace.TracerProvider that you have configured.
var provider trace.TracerProvider

svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.TracerProvider = smithyoteltracing.Adapt(provider)
})
```

SDK operations are instrumented with a span hierarchy that covers the high-level components of the operation lifecycle, such as request serialization, signing, and the retry loop.

Metrics

Use the smithyotelmetrics adapter module to connect an OTel `metric.MeterProvider` to a service client.

The following example shows how to configure metrics for an Amazon S3 client:

```
import (
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/metrics/smithyotelmetrics"
    "go.opentelemetry.io/otel/metric"
)
```

```
// provider is an OTel metric.MeterProvider that you have configured.
var provider metric.MeterProvider

svc := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.MeterProvider = smithyotelmetrics.Adapt(provider)
})
```

Supported metrics

SDK clients collect the following metrics:

Metric name	Unit	Type	Description
client.call.duration	s	Histogram	Overall call duration including retries and time to send or receive request and response body.
client.call.attempts	{attem	Monotonic Counter	The number of attempts for an individual operation.
client.call.errors	{error}	Monotonic Counter	The number of errors for an operation.
client.call.attempt_duration	s	Histogram	The time to connect to the service, send the request, and get back HTTP status code and headers (including time queued waiting to be sent).
client.call.resolve_endpoint_duration	s	Histogram	The time to resolve an endpoint (endpoint resolver, not DNS) for the request.
client.call.deserialization_duration	s	Histogram	The time to deserialize a message body.
client.call.auth.signing_duration	s	Histogram	The time to sign a request.

Metric name	Unit	Type	Description
`client.call.auth.resolve_identity_duration`	s	Histogram	The time to acquire an identity (AWS credentials, bearer token, etc.) from an identity provider.

The following attributes (dimensions) are included with each metric where applicable:

- `rpc.service` – The service name.
- `rpc.method` – The operation name.
- `exception.type` – The error type (included with `client.call.errors`).
- `auth.scheme_id` – The authentication scheme (included with auth-related metrics).

HTTP client metrics

The SDK's HTTP client collects the following additional metrics related to the underlying HTTP connection lifecycle:

Metric name	Unit	Type	Description
`client.http.connections.acquire_duration`	s	Histogram	The time it takes a request to acquire a connection.
`client.http.connections.dns_lookup_duration`	s	Histogram	The time it takes to perform a DNS lookup.
`client.http.connections.tls_handshake_duration`	s	Histogram	The time it takes to perform a TLS handshake.
`client.http.connections.usage`	{conne on}	UpDownCounter	The current state of connections in the pool. Uses a `state` dimension with a value of `idle` or `acquired`.

Metric name	Unit	Type	Description
`client.http.do_request_duration`	s	Histogram	The total time spent performing the HTTP request.
`client.http.time_to_first_byte`	s	Histogram	The time from sending the request to receiving the first response byte.

Retries and Timeouts

The AWS SDK for Go enables you to configure the retry behavior of requests to HTTP services. By default, service clients use retry.Standard as their default retryer. If the default configuration or behavior does not meet your application requirements, you can adjust the retryer configuration or provide your own retryer implementation.

The AWS SDK for Go provides a aws.Retryer interface that defines the set of methods required by a retry implementation to implement. The SDK provides two implementations for retries: retry.Standard and aws.NoOpRetryer.

Standard Retryer

The retry.Standard retryer is the default `aws.Retryer` implementation used by SDK clients. The standard retryer is a rate limited retryer with a configurable number of max attempts, and the ability to tune the request back off policy.

The following table defines the default values for this retryer:

Property	Default
Max Number of Attempts	3
Max Back Off Delay	20 seconds

When a retryable error occurs while invoking your request, the standard retryer will use its provided configuration to delay and subsequently retry the request. Retries add to the overall latency of your request, and you must configure retryer if the default configuration does not meet your application requirements.

See the retry package documentation for details on what errors are considered as retryable by the standard retryer implementation.

NopRetryer

The aws.NopRetryer is a `aws.Retryer` implementation that is provided if you wish to disable all retry attempts. When invoking a service client operation, this retryer will only allow the request to be attempted once, and any resulting error will be returned to the calling application.

Customizing Behavior

The SDK provides a set of helper utilities that wrap an `aws.Retryer` implementation, and returns the provided retryer wrapped with the desired retry behavior. You can override the default retryer for all clients, per client, or per operation depending on your applications requirements. To see additional examples showing how to do this, see the retry package documentation examples.

> **Warning**
>
> If specifying a global `aws.Retryer` implementation using `config.WithRetryer`, you must ensure that you return a new instance of the `aws.Retryer` each invocation. This will ensure that you won't create a global retry token bucket across all service clients.

Limiting the max number of attempts

You use retry.AddWithMaxAttempts to wrap an `aws.Retryer` implementation to set the max number attempts to your desired value.

> **Warning**
>
> Using `retry.AddWithMaxAttempts` with a value of zero will allow the SDK to retry all retryable errors until the request succeeds, or a non-retryable error is returned. **Allowing the SDK to retry infinitely may result in runaway workloads and inflated billing cycles.**

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
```

```
// ...

// MaxAttempts will be infinite (will retry indefinitely)
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithMaxAttempts(retry.NewStandard(), 0)
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Note that instantiating a retryer using functional options to set MaxAttempts directly will have slightly different behavior. More specifically, setting a value less than or equal to zero will cause the retryer to use the default maximum of 3 attempts, rather than retry infinitely:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

// MaxAttempts will default to 3
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        o.MaxAttempts = 0
    })
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Limiting the max back off delay

You use retry.AddWithMaxBackoffDelay to wrap an `aws.Retryer` implementation and limit the max back off delay that is allowed to occur between retrying a failed request.

For example, you can the following code to wrap the standard client retryer with a desired max delay of five seconds:

```
import "context"
import "time"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithMaxBackoffDelay(retry.NewStandard(), time.Second*5)
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Retry additional API error codes

You use retry.AddWithErrorCodes to wrap an `aws.Retryer` implementation and include additional API error codes that should be considered retryable.

For example, you can the following code to wrap the standard client retryer to include the Amazon S3 `NoSuchBucketException` exception as retryable.

```
import "context"
import "time"
import "github.com/aws/aws-sdk-go-v2/aws/retry"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRetryer(func()
 aws.Retryer {
    return retry.AddWithErrorCodes(retry.NewStandard(), (*types.NoSuchBucketException)
(nil).ErrorCode())
```

```
}))
if err != nil {
    return err
}

client := s3.NewFromConfig(cfg)
```

Client-side rate limiting

The AWS SDK for Go introduces a new client-side rate-limiting mechanism in the standard retry policy to align with the behavior of modern SDKs. This is behavior is controlled by the RateLimiter field on a retryer's options.

A RateLimiter operates as a token bucket with a set capacity, where operation attempt failures consume tokens. A retry that attempts to consume more tokens than what's available results in operation failure with a QuotaExceededError.

The default implementation is parameterized as follows (how to modify each setting):

- a capacity of 500 (set the value of RateLimiter on StandardOptions using NewTokenRateLimit)
- a retry caused by a timeout costs 10 tokens (set RetryTimeoutCost on StandardOptions)
- a retry caused by other errors costs 5 tokens (set RetryCost on StandardOptions)
- an operation that succeeds on the 1st attempt adds 1 token (set NoRetryIncrement on StandardOptions)
  - operations that succeed on the 2nd or later attempt do not add back any tokens

If you find that the default behavior does not fit your application's needs, you can disable it with ratelimit.None.

Example: modified rate limiter

```
import (
    "context"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/aws/ratelimit"
    "github.com/aws/aws-sdk-go-v2/aws/retry"
    "github.com/aws/aws-sdk-go-v2/config"
)
```

```
// ...

cfg, err := config.LoadDefaultConfig(context.Background(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        // Makes the rate limiter more permissive in general. These values are
        // arbitrary for demonstration and may not suit your specific
        // application's needs.
        o.RateLimiter = ratelimit.NewTokenRateLimit(1000)
        o.RetryCost = 1
        o.RetryTimeoutCost = 3
        o.NoRetryIncrement = 10
    })
}))
```

Example: no rate limit using ratelimit.None

```
import (
    "context"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/aws/ratelimit"
    "github.com/aws/aws-sdk-go-v2/aws/retry"
    "github.com/aws/aws-sdk-go-v2/config"
)

// ...

cfg, err := config.LoadDefaultConfig(context.Background(), config.WithRetryer(func()
 aws.Retryer {
    return retry.NewStandard(func(o *retry.StandardOptions) {
        o.RateLimiter = ratelimit.None
    })
}))
```

Timeouts

You use the context package to set timeouts or deadlines when invoking a service client operation. Use the context.WithDeadline to wrap your applications context and set a deadline to a specific time by which the invoked operation must be completed. To set a timeout after a certain `time.Duration` use context.WithTimeout. The SDK passes the provided `context.Context` to the HTTP transport client when invoking a service API. If the context passed to the SDK is cancelled

or becomes cancelled while invoking the operation, the SDK will not retry the request further and will return to the calling application. You must handle context cancellation appropriately in your application in cases where the context provided to the SDK has become cancelled.

Setting a timeout

The following example shows how to set a timeout for a service client operation.

```
import "context"
import "time"

// ...

ctx := context.TODO() // or appropriate context.Context value for your application

client := s3.NewFromConfig(cfg)

// create a new context from the previous ctx with a timeout, e.g. 5 seconds
ctx, cancel := context.WithTimeout(ctx, 5*time.Second)
defer cancel()

resp, err := client.GetObject(ctx, &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    // handle error
}
```

Using the AWS SDK for Go

Learn common and recommended ways of programming with the AWS SDK for Go in your applications.

Topics

- Constructing a Service Client
- Calling Service Operations
- Concurrently Using Service Clients
- Using Operation Paginators
- Using Waiters
- Handling Errors in the AWS SDK for Go V2

Constructing a Service Client

Service clients can be constructed using either the `New` or `NewFromConfig` functions available in service client's Go package. Each function will return a `Client` struct type containing the methods for invoking the service APIs. The `New` and `NewFromConfig` each provide the same set of configurable options for constructing a service client, but provide slightly different construction patterns that we will look at in the following sections.

NewFromConfig

`NewFromConfig` function provides a consistent interface for constructing service clients using the aws.Config. An `aws.Config` can be loaded using the config.LoadDefaultConfig. For more information on constructing an `aws.Config`, see Configure the SDK. The following example shows how to construct an Amazon S3 service client using the `aws.Config`and the `NewFromConfig` function:

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...
```

```
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg)
```

Overriding Configuration

`NewFromConfig` can take one or more functional arguments that can mutate a client's configuration `Options` struct. This allows you to make specific overrides such as changing the Region, or modifying service specific options such as Amazon S3 `UseAccelerate` option. For example:

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg, func(o *s3.Options) {
        o.Region = "us-west-2"
        o.UseAccelerate = true
    })
```

Overrides to the client `Options` value is determined by the order that the functional arguments are given to `NewFromConfig`.

New

> **Note**
>
> New is considered a more advanced form of client construction. We recommend you use `NewFromConfig` for client construction, as it allows construction using the `aws.Config`

struct. This removes the need to construct an `Options` struct instance for each service client your application requires.

New function is a client constructor provides an interface for constructing clients using only the client packages `Options` struct for defining the client's configuration options. For example, to construct Amazon S3 client using New:

```
import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/credentials"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    client := s3.New(s3.Options{
        Region:      "us-west-2",
        Credentials:
 aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey, secretKey,
 "")),
    })
```

Overriding Configuration

New can take one or more functional arguments that can mutate a client's configuration `Options` struct. This allows you to make specific overrides such as changing the Region or modifying service specific options such as Amazon S3 `UseAccelerate` option. For example:

```
import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/credentials"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    options := s3.Options{
        Region:      "us-west-2",
        Credentials:
 aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey, secretKey,
 "")),
    }
```

```
    client := s3.New(options, func(o *s3.Options) {
        o.Region = "us-east-1"
        o.UseAccelerate = true
    })
```

Overrides to the client `Options` value is determined by the order that the functional arguments are given to `New`.

Calling Service Operations

After you have a service client instance, you can use it to call a service's operations. For example, to call the Amazon S3 `GetObject` operation:

```
response, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("obj-key"),
    })
```

When you call a service operation, the SDK synchronously validates the input, serializes the request, signs it with your credentials, sends it to AWS, and then deserializes a response or an error. In most cases, you can call service operations directly. Each service operation client method will return an operation response struct, and an error interface type. You should always check `error` type to determine if an error occurred before attempting to access the service operation's response struct.

Passing Parameters to a Service Operation

Each service operation method takes a context.Context value that can be used for setting request deadlines that will be honored by the SDK. In addition, each service operation will take a `<OperationName>Input` struct found in the service's respective Go package. You pass in API input parameters using the operation input struct.

Operation input structures can have input parameters such as the standard Go numerics, boolean, string, map, and list types. In more complex API operations a service might have more complex modeling of input parameters. These other types such as service specific structures and enum values are found in the service's `types` Go package.

In addition, services might distinguish between the default value of a Go type and whether the value was set or not by the user. In these cases, input parameters might require you to pass a pointer reference to the type in question. For standard Go types like numerics, boolean, and string there are `<Type>` and `From<Type>` convenience functions available in the aws to ease this conversion. For example, aws.String can be used to convert a `string` to a `*string` type for input parameters that require a pointer to a string. Inversely, aws.ToString can be used to transform a `*string` to a `string` while providing protection from dereferencing a nil pointer. The `To<Type>` functions are helpful when handling service responses.

Let's look at an example of how we can use an Amazon S3 client to call the `GetObject` API, and construct our input using the `types` package, and `aws.<Type>` helpers.

```
import "context"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"
    import "github.com/aws/aws-sdk-go-v2/service/s3/types"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        panic(err)
    }

    client := s3.NewFromConfig(cfg)

    resp, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        Bucket:       aws.String("amzn-s3-demo-bucket"),
        Key:          aws.String("keyName"),
        RequestPayer: types.RequestPayerRequester,
    })
```

Overriding Client Options For Operation Call

Similar to how client operation options can be modified during construction of a client using functional arguments, the client options can be modified at the time the operation method is called by providing one or more functional arguments to the service operation method. This action is concurrency safe and will not affect other concurrent operations on the client.

For example, to override the client region from "us-west-2" to "us-east-1":

```
cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithRegion("us-west-2"))
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := s3.NewFromConfig(cfg)

    params := &s3.GetObjectInput{
        // ...
    }

    resp, err := client.GetObject(context.TODO(), params, func(o *Options) {
        o.Region = "us-east-1"
    })
```

Handling Operation Responses

Each service operation has an associated output struct that contains the service's operation response members. The output struct follows the following naming pattern `<OperationName>Output`. Some operations might have no members defined for their operation output. After calling a service operation, the return `error` argument type should always be checked to determine if an error occurred while invoking the service operation. Errors returned can range from client-side input validation errors to service-side error responses returned to the client. The operation's output struct should not be accessed in the event that a non-nil error is returned by the client.

For example, to log an operation error and prematurely return from the calling function:

```
response, err := client.GetObject(context.TODO())
    if err != nil {
        log.Printf("GetObject error: %v", err)
        return
    }
```

For more information on error handling, including how to inspect for specific error types, see TODO

Responses with `io.ReadCloser`

Some API operations return a response struct that contain an output member that is an `io.ReadCloser`. This will be the case for API operations that expose some element of their output in the body of the HTTP response itself.

For example, Amazon S3 `GetObject` operation returns a response whose `Body` member is an `io.ReadCloser` for accessing the object payload.

> **Warning**
>
> You MUST ALWAYS `Close()` any `io.ReadCloser` output members, regardless of whether you have consumed its content. Failure to do so can leak resources and potentially create issues with reading response bodies for operations called in the future.

```
resp, err := s3svc.GetObject(context.TODO(), &s3.GetObjectInput{...})
    if err != nil {
        // handle error
        return
    }
    // Make sure to always close the response Body when finished
    defer resp.Body.Close()

    decoder := json.NewDecoder(resp.Body)
    if err := decoder.Decode(&myStruct); err != nil {
        // handle error
        return
    }
```

Response Metadata

All service operation output structs include a `ResultMetadata` member of type middleware.Metadata. `middleware.Metadata` is used by the SDK middleware to provide additional information from a service response that is not modeled by the service. This includes metadata like the `RequestID`. For example, to retrieve the `RequestID` associated with a service response to assist AWS Support in troubleshooting a request:

```
import "fmt"
```

```
    import "log"
    import "github.com/aws/aws-sdk-go-v2/aws/middleware"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ..

    resp, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
        // ...
    })
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    requestID, ok := middleware.GetRequestIDMetadata(resp.ResultMetadata)
    if !ok {
        fmt.Println("RequestID not included with request")
    }

    fmt.Printf("RequestID: %s\n", requestID)
```

Concurrently Using Service Clients

You can create goroutines that concurrently use the same service client to send multiple requests. You can use a service client with as many goroutines as you want.

In the following example, an Amazon S3 service client is used in multiple goroutines. This example concurrently uploads two objects to an Amazon S3 bucket.

```
import "context"
    import "log"
    import "strings"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
```

```
    }

    client := s3.NewFromConfig(cfg)

    type result struct {
        Output *s3.PutObjectOutput
        Err    error
    }

    results := make(chan result, 2)

    var wg sync.WaitGroup
    wg.Add(2)

    go func() {
    defer wg.Done()
        output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
            Bucket: aws.String("amzn-s3-demo-bucket"),
            Key:    aws.String("foo"),
            Body:   strings.NewReader("foo body content"),
        })
        results <- result{Output: output, Err: err}
    }()

    go func() {
        defer wg.Done()
        output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
            Bucket: aws.String("amzn-s3-demo-bucket"),
            Key:    aws.String("bar"),
            Body:   strings.NewReader("bar body content"),
        })
        results <- result{Output: output, Err: err}
    }()

    wg.Wait()

    close(results)

    for result := range results {
        if result.Err != nil {
            log.Printf("error: %v", result.Err)
            continue
        }
        fmt.Printf("etag: %v", aws.ToString(result.Output.ETag))
```

```
    }
```

Using Operation Paginators

Typically, when you retrieve a list of items, you might need to check the output struct for a token or marker to confirm whether the AWS service returned all results from your request. If the token or marker is present, you use it to request the next page of results. Instead of managing these tokens or markers, you can use the service package's available paginator types.

Paginator helpers are available for supported service operations, and can be found in the service client's Go package. To construct a paginator for a supported operation, use the `New<OperationName>Paginator` function. Paginator construct functions take the service `Client`, the operation's `<OperationName>Input` input parameters, and an optional set of functional arguments allowing you to configure other optional paginator settings.

The returned operation paginator type provides a convenient way to iterate over a paginated operation until you have reached the last page, or you have found the item(s) that your application was searching for. A paginator type has two methods: `HasMorePages` and `NextPage`. `HasMorePages` returns a boolean value of `true` if the first page has not been retrieved, or if additional pages available to retrieve using the operation. To retrieve the first or subsequent pages of the operation, the `NextPage` operation must be called. `NextPage` takes `context.Context` and returns the operation output and any corresponding error. Like the client operation method return parameters, the return error should always be checked before attempting to use the returned response structure. See Handling Operation Responses.

The following example uses the `ListObjectsV2` paginator to list up to three pages of object keys from the `ListObjectV2`operation. Each page consists of up to 10 keys, which is defined by the `Limit` paginator option.

```
import "context"
    import "log"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/service/s3"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
```

```
        log.Printf("error: %v", err)
        return
    }

    client := s3.NewFromConfig(cfg)

    params := &s3.ListObjectsV2Input{
        Bucket: aws.String("amzn-s3-demo-bucket"),
    }

    paginator := s3.NewListObjectsV2Paginator(client, params, func(o
*s3.ListObjectsV2PaginatorOptions) {
        o.Limit = 10
    })

    pageNum := 0
    for paginator.HasMorePages() && pageNum < 3 {
        output, err := paginator.NextPage(context.TODO())
        if err != nil {
            log.Printf("error: %v", err)
            return
        }
        for _, value := range output.Contents {
            fmt.Println(*value.Key)
        }
        pageNum++
    }
```

Similar to client operation method, the client options like the request Region can be modified by providing one or more functional arguments to `NextPage`. For more information about overriding client options when calling an operation, see Overriding Client Options For Operation Call.

Using Waiters

When interacting with AWS APIs that are asynchronous, you often need to wait for a particular resource to become available in order to perform further actions on it.

For example, the Amazon DynamoDB `CreateTable` API returns immediately with a TableStatus of CREATING, and you can't invoke read or write operations until the table status has been transitioned to `ACTIVE`.

Writing logic to continuously poll the table status can be cumbersome and error-prone. The waiters help take the complexity out of it and are simple APIs that handle the polling task for you.

For example, you can use waiters to poll if a DynamoDB table is created and ready for a write operation.

```
import "context"
    import "fmt"
    import "log"
    import "time"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client)

    // params is the input to api operation used by the waiter
    params := &dynamodb.DescribeTableInput {
        TableName: aws.String("test-table")
    }

    // maxWaitTime is the maximum wait time, the waiter will wait for
    // the resource status.
    maxWaitTime := 5 * time.Minutes

    // Wait will poll until it gets the resource status, or max wait time
    // expires.
    err := waiter.Wait(context.TODO(), params, maxWaitTime)
    if err != nil {
        log.Printf("error: %v", err)
        return
```

```
    }
    fmt.Println("Dynamodb table is now ready for write operations")
```

Overriding waiter configuration

By default, the SDK uses the minimum delay and maximum delay value configured with optimal values defined by AWS services for different APIs. You can override waiter configuration by providing functional options during waiter construction, or when invoking a waiter operation.

For example, to override waiter configuration during waiter construction

```
import "context"
    import "fmt"
    import "log"
    import "time"
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"

    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client, func (o
 *dynamodb.TableExistsWaiterOptions) {

        // override minimum delay to 10 seconds
        o.MinDelay = 10 * time.Second

        // override maximum default delay to 300 seconds
        o.MaxDelay = 300 * time.Second
    })
```

The `Wait` function on each waiter also takes in functional options. Similar to the above example, you can override waiter configuration per `Wait` request.

```
// params is the input to api operation used by the waiter
    params := &dynamodb.DescribeTableInput {
        TableName: aws.String("test-table")
    }

    // maxWaitTime is the maximum wait time, the waiter will wait for
    // the resource status.
    maxWaitTime := 5 * time.Minutes

    // Wait will poll until it gets the resource status, or max wait time
    // expires.
    err := waiter.Wait(context.TODO(), params, maxWaitTime, func (o
 *dynamodb.TableExistsWaiterOptions) {

        // override minimum delay to 5 seconds
        o.MinDelay = 5 * time.Second

        // override maximum default delay to 120 seconds
        o.MaxDelay = 120 * time.Second
    })
    if err != nil {
        log.Printf("error: %v", err)
        return
    }
    fmt.Println("Dynamodb table is now ready for write operations")
```

Advanced waiter configuration overrides

You can additionally customize the waiter default behavior by providing a custom retryable function. The waiter-specific options also provides `APIOptions` to customize operation middlewares.

For example, to configure advanced waiter overrides.

```
import "context"
    import "fmt"
    import "log"
    import "time"
```

```
    import "github.com/aws/aws-sdk-go-v2/aws"
    import "github.com/aws/aws-sdk-go-v2/config"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb"
    import "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
    // ...

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Printf("error: %v", err)
        return
    }

    client := dynamodb.NewFromConfig(cfg)

    // custom retryable defines if a waiter state is retryable or a terminal state.
    // For example purposes, we will configure the waiter to not wait
    // if table status is returned as `UPDATING`
    customRetryable := func(ctx context.Context, params *dynamodb.DescribeTableInput,
        output *dynamodb.DescribeTableOutput, err error) (bool, error) {
        if output.Table != nil {
            if output.Table.TableStatus == types.TableStatusUpdating {
                // if table status is `UPDATING`, no need to wait
                return false, nil
            }
        }
    }

    // we create a waiter instance by directly passing in a client
    // that satisfies the waiters client Interface.
    waiter :=  dynamodb.NewTableExistsWaiter(client, func (o
*dynamodb.TableExistsWaiterOptions) {

        // override the service defined waiter-behavior
        o.Retryable = customRetryable
    })
```

Handling Errors in the AWS SDK for Go V2

The AWS SDK for Go returns errors that satisfy the Go `error` interface type. You can use the `Error()` method to get a formatted string of the SDK error message without any special handling. Errors returned by the SDK may implement an `Unwrap` method. The `Unwrap` method is used

by the SDK to provide additional contextual information to errors, while providing access to the underlying error or chain of errors. The `Unwrap` method should be used with the errors.As to handle unwrapping error chains.

It is important that your application check whether an error occurred after invoking a function or method that can return an `error` interface type. The most basic form of error handling looks similar to the following example:

```
if err != nil {
    // Handle error
    return
}
```

Logging Errors

The simplest form of error handling is traditionally to log or print the error message before returning or exiting from the application.

```
import "log"

// ...

if err != nil {
    log.Printf("error: %s", err.Error())
    return
}
```

Service Client Errors

The SDK wraps all errors returned by service clients with the smithy.OperationError error type. `OperationError` provides contextual information about the service name and operation that is associated with an underlying error. This information can be useful for applications that perform batches of operations to one or more services, with a centralized error handling mechanism. Your application can use `errors.As` to access this `OperationError` metadata.

```
import "log"
import "github.com/aws/smithy-go"

// ...
```

```
if err != nil {
    var oe *smithy.OperationError
    if errors.As(err, &oe) {
        log.Printf("failed to call service: %s, operation: %s, error: %v",
 oe.Service(), oe.Operation(), oe.Unwrap())
    }
    return
}
```

API Error Responses

Service operations can return modeled error types to indicate specific errors. These modeled types can be used with `errors.As` to unwrap and determine if the operation failure was due to a specific error. For example, Amazon S3 `CreateBucket` can return a BucketAlreadyExists error if a bucket of the same name already exists.

For example, to check if an error was a `BucketAlreadyExists` error:

```
import "log"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"

// ...

if err != nil {
    var bne *types.BucketAlreadyExists
    if errors.As(err, &bne) {
        log.Println("error:", bne)
    }
    return
}
```

All service API response errors implement the smithy.APIError interface type. This interface can be used to handle both modeled or un-modeled service error responses. This type provides access to the error code and message returned by the service. Additionally, this type provides indication of whether the fault of the error was due to the client or server if known.

```
import "log"
import "github.com/aws/smithy-go"

// ...
```

```
if err != nil {
    var ae smithy.APIError
    if errors.As(err, &ae) {
        log.Printf("code: %s, message: %s, fault: %s", ae.ErrorCode(),
 ae.ErrorMessage(), ae.ErrorFault().String())
    }
    return
}
```

Retrieving Request Identifiers

When working with AWS Support, you may be asked to provide the request identifier that identifies the request you are attempting to troubleshoot. You can use http.ResponseError and use the `ServiceRequestID()` method to retrieve the request identifier associated with error response.

```
import "log"
import awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"

// ...

if err != nil {
    var re *awshttp.ResponseError
    if errors.As(err, &re) {
        log.Printf("requestID: %s, error: %v", re.ServiceRequestID(), re.Unwrap());
    }
    return
}
```

Amazon S3 Request Identifiers

Amazon S3 requests contain additional identifiers that can be used to assist AWS Support with troubleshooting your request. You can use s3.ResponseError and call `ServiceRequestID()` and `ServiceHostID()` to retrieve the request ID and host ID.

```
import "log"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

if err != nil {
    var re s3.ResponseError
```

```
    if errors.As(err, &re) {
        log.Printf("requestID: %s, hostID: %s request failure", re.ServiceRequestID(),
 re.ServiceHostID());
    }
    return
}
```

Use the AWS SDK for Go v2 with AWS services

To make calls to an AWS service, you must first construct a service client instance. A service client provides low-level access to every API action for that service. For example, you create an Amazon S3 service client to make calls to Amazon S3 APIs.

When you call service operations, you pass in input parameters as a struct. A successful call will result in an output struct containing the service API response. For example, after you successfully call an Amazon S3 create bucket action, the action returns an output struct with the bucket's location.

For the list of service clients, including their methods and parameters, see the AWS SDK for Go API Reference.

Data integrity protection with checksums

Amazon Simple Storage Service (Amazon S3) provides the ability to specify a checksum when you upload an object. When you specify a checksum, it is stored with the object and can be validated when the object is downloaded.

Checksums provide an additional layer of data integrity when you transfer files. With checksums, you can verify data consistency by confirming that the received file matches the original file. For more information about checksums with Amazon S3, see the Amazon Simple Storage Service User Guide including the supported algorithms.

You have the flexibility to choose the algorithm that best fits your needs and let the SDK calculate the checksum. Alternatively, you can provide a pre-computed checksum value by using one of the supported algorithms.

> **Note**
>
> Beginning with version v1.74.1 of the Amazon S3 module, the SDK provides default integrity protections by automatically calculating a CRC32 checksum for uploads. The SDK calculates this checksum if you don't provide a precalculated checksum value or if you don't specify an algorithm that the SDK should use to calculate a checksum.
> The SDK also provides global settings for data integrity protections that you can set externally, which you can read about in the AWS SDKs and Tools Reference Guide.

We discuss checksums in two request phases: uploading an object and downloading an object.

Upload an object

When you upload an object with the `putObject` method and provide a checksum algorithm, the SDK computes the checksum for the specified algorithm.

The following code snippet shows a request to upload an object with a CRC32 checksum. When the SDK sends the request, it calculates the CRC32 checksum and uploads the object. Amazon S3 validates the integrity of the content by calculating the checksum and comparing it to the checksum provided by the SDK. Amazon S3 then stores the checksum with the object.

```
out, err := s3Client.PutObject(context.Background(), &s3.PutObjectInput{
     Bucket:            aws.String("bucket"),
     Key:               aws.String("key"),
     ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
     Body:              strings.NewReader("Hello World"),
 })
```

If you don't provide a checksum algorithm with the request, the checksum behavior varies depending on the version of the SDK that you use as shown in the following table.

Checksum behavior when no checksum algorithm is provided

Amazon S3 module version of AWS SDK for Go	Checksum behavior
Earlier than v1.74.1	The SDK doesn't automatically calculate a CRC-based checksum and provide it in the request.
v1.74.1 or later	The SDK uses the CRC32 algorithm to calculate the checksum and provides it in the request. Amazon S3 validates the integrity of the transfer by computing its own CRC32 checksum and compares it to the checksum provided by the SDK. If the checksums match, the checksum is saved with the object.

Use a pre-calculated checksum value

A pre-calculated checksum value provided with the request disables automatic computation by the SDK and uses the provided value instead.

The following example shows a request with a pre-calculated SHA256 checksum.

```
out, err := s3Client.PutObject(context.Background(), &s3.PutObjectInput{
     Bucket:        aws.String("bucket"),
     Key:           aws.String("key"),
     ChecksumCRC32: aws.String("checksumvalue"),
     Body:          strings.NewReader("Hello World"),
 })
```

If Amazon S3 determines the checksum value is incorrect for the specified algorithm, the service returns an error response.

Multipart uploads

You can also use checksums with multipart uploads.

The AWS SDK for Go provides two options to use checksums with multipart uploads. The first option uses the transfer manager that specifies the CRC32 algorithm for the upload.

```
s3Client := s3.NewFromConfig(cfg)
     transferManager := manager.NewUploader(s3Client)
     out, err := transferManager.Upload(context.Background(), &s3.PutObjectInput{
             Bucket:            aws.String("bucket"),
             Key:               aws.String("key"),
             Body:              large file to trigger multipart upload,
             ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
     })
```

If you don't provide a checksum algorithm when using the transfer manager for uploads, the SDK automatically calculates and checksum based on the CRC32 algorithm. The SDK performs this calculation for all versions of the SDK.

The second option uses the Amazon S3 client to perform the multipart upload. If you specify a checksum with this approach, you must specify the algorithm to use on the initiation of the upload. You must also specify the algorithm for each part request and provide the checksum calculated for each part after it is uploaded.

```
s3Client := s3.NewFromConfig(cfg)
    createMultipartUploadOutput, err :=
 s3Client.CreateMultipartUpload(context.Background(), &s3.CreateMultipartUploadInput{
            Bucket:            aws.String("bucket"),
            Key:               aws.String("key"),
            ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
         })
    if err != nil {
            log.Fatal("err create multipart upload ", err)
    }

    var partsBody []io.Reader // this is just an example parts content, you should
 load your target file in your code
    partNum := int32(1)
    var completedParts []types.CompletedPart
    for _, body := range partsBody {
        uploadPartOutput, err := s3Client.UploadPart(context.Background(),
 &s3.UploadPartInput{
            Bucket:            aws.String("bucket"),
            Key:               aws.String("key"),
            ChecksumAlgorithm: types.ChecksumAlgorithmCrc32,
            Body:              body,
            PartNumber:        aws.Int32(partNum),
            UploadId:          createMultipartUploadOutput.UploadId,
        })
        if err != nil {
            log.Fatal("err upload part ", err)
        }

        completedParts = append(completedParts, types.CompletedPart{
            PartNumber:    aws.Int32(partNum),
            ETag:          uploadPartOutput.ETag,
            ChecksumCRC32: uploadPartOutput.ChecksumCRC32,
        })
        partNum++
    }

    completeMultipartUploadOutput, err :=
 s3Client.CompleteMultipartUpload(context.Background(),
 &s3.CompleteMultipartUploadInput{
        Bucket:   aws.String("bucket"),
        Key:      aws.String("key"),
        UploadId: createMultipartUploadOutput.UploadId,
```

```
        MultipartUpload: &types.CompletedMultipartUpload{
            Parts: completedParts,
        },
    })
    if err != nil {
        log.Fatal("err complete multipart upload ", err)
    }
```

Download an object

When you use the GetObject method to download an object, the SDK automatically validates the checksum when the `ChecksumMode` field of `GetObjectInput` is set to `types.ChecksumModeEnabled`.

The request in the following snippet directs the SDK to validate the checksum in the response by calculating the checksum and comparing the values.

```
out, err := s3Client.GetObject(context.Background(), &s3.GetObjectInput{
    Bucket:       aws.String("bucket"),
    Key:          aws.String("key"),
    ChecksumMode: types.ChecksumModeEnabled,
 })
```

If the object wasn't uploaded with a checksum, no validation takes place.

Using the AWS SDK for Go Utilities

The AWS SDK for Go includes the following utilities to help you more easily use AWS services. Find the SDK utilities in their related AWS service package.

Amazon RDS Utilities

IAM Authentication

The auth package provides utilities for generating authentication tokens for connecting to Amazon RDS MySQL and PostgreSQL database instances. Using the BuildAuthToken method, you generate a database authorization token by providing the database endpoint, AWS Region, username, and a aws.CredentialProvider implementation that returns IAM credentials with permission to connect to the database using IAM database authentication. To learn more about configuring Amazon RDS with IAM authentication, see the following Amazon RDS Developer Guide resources:

- Enabling and disabling IAM database authentication
- Creating and using an IAM policy for IAM database access
- Creating a database account using IAM authentication

The following example shows how to generate an authentication token to connect to an Amazon RDS database:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/feature/rds/auth"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    panic("configuration error: " + err.Error())
}

authenticationToken, err := auth.BuildAuthToken(
    context.TODO(),
    "mydb.123456789012.us-east-1.rds.amazonaws.com:3306", // Database Endpoint (With
 Port)
```

```
    "us-east-1", // AWS Region
    "jane_doe", // Database Account
    cfg.Credentials,
)
if err != nil {
    panic("failed to create authentication token: " + err.Error())
}
```

Amazon CloudFront Utilities

Amazon CloudFront URL Signer

The Amazon CloudFront URL signer simplifies the process of creating signed URLs. A signed URL includes information, such as an expiration date and time, that enables you to control access to your content. Signed URLs are useful when you want to distribute content through the internet, but want to restrict access to certain users (for example, to users who have paid a fee).

To sign a URL, create a `URLSigner` instance with your CloudFront key pair ID and the associated private key. Then call the `Sign` or `SignWithPolicy` method and include the URL to sign. For more information about Amazon CloudFront key pairs, see Creating CloudFront Key Pairs for Your Trusted Signers in the CloudFront Developer Guide.

The following example creates a signed URL that's valid for one hour after it is created.

```
import "github.com/aws/aws-sdk-go-v2/feature/cloudfront/sign"

// ...

signer := sign.NewURLSigner(keyID, privKey)

signedURL, err := signer.Sign(rawURL, time.Now().Add(1*time.Hour))
if err != nil {
    log.Fatalf("Failed to sign url, err: %s\n", err.Error())
    return
}
```

For more information about the signing utility, see the sign package in the AWS SDK for Go API Reference.

Amazon EC2 Instance Metadata Service

You can use the AWS SDK for Go to access the Amazon EC2 Instance Metadata Service. The feature/ec2/imds Go package provides a Client type that can be used to access the Amazon EC2 Instance Metadata Service. The `Client` and associated operations can be used similar to the other AWS service clients provided by the SDK. To learn more information on how to configure the SDK, and use service clients see Configure the SDK and Use the AWS SDK for Go v2 with AWS services.

The client can help you easily retrieve information about instances on which your applications run, such as its AWS Region or local IP address. Typically, you must create and submit HTTP requests to retrieve instance metadata. Instead, create an `imds.Client` to access the Amazon EC2 Instance Metadata Service using a programmatic client like other AWS Services.

For example to construct a client:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/feature/ec2/imds"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Printf("error: %v", err)
    return
}

client := imds.NewFromConfig(cfg)
```

Then use the service client to retrieve information from a metadata category such as `local-ipv4` (the private IP address of the instance).

```
localIp, err := client.GetMetadata(context.TODO(), &imds.GetMetadataInput{
    Path: "local-ipv4",
})
if err != nil {
    log.Printf("Unable to retrieve the private IP address from the EC2 instance: %s\n",
 err)
    return
}
content, _ := io.ReadAll(localIp.Content)
```

```
fmt.Printf("local-ip: %v\n", string(content))
```

For a list of all metadata categories, see Instance metadata categories in the Amazon EC2 User Guide.

Amazon S3 Utilities

Amazon S3 Transfer Managers

The Amazon S3 upload and download managers can break up large objects, so they can be transferred in multiple parts, in parallel. This makes it easy to resume interrupted transfers.

Amazon S3 Upload Manager

The Amazon S3 upload manager determines if a file can be split into smaller parts and uploaded in parallel. You can customize the number of parallel uploads and the size of the uploaded parts.

The following example uses the Amazon S3 `Uploader` to upload a file. Using `Uploader` is similar to the `s3.PutObject()` operation.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/feature/s3/manager"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Printf("error: %v", err)
    return
}

client := s3.NewFromConfig(cfg)

uploader := manager.NewUploader(client)
result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-object-key"),
    Body:   uploadFile,
})
```

Configuration Options

When you instantiate an `Uploader` instance using NewUploader, you can specify several configuration options to customize how objects are uploaded. Options are overridden by providing one or more arguments to `NewUploader`. These options include:

- `PartSize` – Specifies the buffer size, in bytes, of each part to upload. The minimum size per part is 5 MiB.
- `Concurrency` – Specifies the number of parts to upload in parallel.
- `LeavePartsOnError` – Indicates whether to leave successfully uploaded parts in Amazon S3.

The `Concurrency` value limits the concurrent number of part uploads that can occur for a given `Upload` call. This is not a global client concurrency limit. Tweak the `PartSize` and `Concurrency` configuration values to find the optimal configuration. For example, systems with high-bandwidth connections can send bigger parts and more uploads in parallel.

For example, your application configures `Uploader` with a `Concurrency` of setting of `5`. If your application then calls `Upload` from two different goroutines, the result is `10` concurrent part uploads (2 goroutines * 5 `Concurrency`).

> **Warning**
>
> Your application is expected to limit the concurrent calls to `Upload` to prevent application resource exhaustion.

Below is an example to set the default part size during `Uploader` creation:

```
uploader := manager.NewUploader(client, func(u *Uploader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

For more information about `Uploader` and its configurations, see Uploader in the AWS SDK for Go API Reference.

PutObjectInput Body Field (io.ReadSeeker vs. io.Reader)

The Body field of the `s3.PutObjectInput` struct is an `io.Reader` type. However, this field can be populated with a type that satisfies both the `io.ReadSeeker` and `io.ReaderAt` interface to

improve application resource utilization of the host environment. The following example creates the type `ReadSeekerAt` that satisfies both interfaces:

```
type ReadSeekerAt interface {
    io.ReadSeeker
    io.ReaderAt
}
```

For `io.Reader` types, the bytes of the reader must be buffered in memory before the part can be uploaded. When you increase the `PartSize` or `Concurrency` value, the required memory (RAM) for the `Uploader` increases significantly. The required memory is approximately *`PartSize`* * *`Concurrency`*. For example, specifying 100 MB for `PartSize` and 10 for `Concurrency`, requires at least 1 GB.

Because an `io.Reader` type cannot determine its size before reading its bytes, `Uploader` cannot calculate how many parts will be uploaded. Consequently, `Uploader` can reach the Amazon S3 upload limit of 10,000 parts for large files if you set the `PartSize` too low. If you try to upload more than 10,000 parts, the upload stops and returns an error.

For body values that implement the `ReadSeekerAt` type, the `Uploader` doesn't buffer the body contents in memory before sending it to Amazon S3. `Uploader` calculates the expected number of parts before uploading the file to Amazon S3. If the current value of `PartSize` requires more than 10,000 parts to upload the file, `Uploader` increases the part size value so that fewer parts are required.

Handling Failed Uploads

If an upload to Amazon S3 fails, by default, `Uploader` uses the Amazon S3 `AbortMultipartUpload` operation to remove the uploaded parts. This functionality ensures that failed uploads do not consume Amazon S3 storage.

You can set `LeavePartsOnError` to true so that the `Uploader` doesn't delete successfully uploaded parts. This is useful for resuming partially completed uploads. To operate on uploaded parts, you must get the `UploadID` of the failed upload. The following example demonstrates how to use the `manager.MultiUploadFailure` error interface type to get the `UploadID`.

```
result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-object-key"),
    Body:   uploadFile,
```

```
})
output, err := u.upload(input)
if err != nil {
    var mu manager.MultiUploadFailure
    if errors.As(err, &mu) {
        // Process error and its associated uploadID
        fmt.Println("Error:", mu)
        _ = mu.UploadID() // retrieve the associated UploadID
    } else {
        // Process error generically
        fmt.Println("Error:", err.Error())
    }
    return
}
```

Overriding Uploader Options Per Upload

You can override the `Uploader` options when calling `Upload` by providing one or more arguments to the method. These overrides are concurrency-safe modifications and do not affect ongoing uploads, or subsequent `Upload` calls to the manager. For example, to override the `PartSize` configuration for a specific upload request:

```
params := &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
    Body:   myBody,
}
resp, err := uploader.Upload(context.TODO(), params, func(u *manager.Uploader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

Examples

Upload a Folder to Amazon S3

The following example uses the `path/filepath` package to recursively gather a list of files and upload them to the specified Amazon S3 bucket. The keys of the Amazon S3 objects are prefixed with the file's relative path.

```
package main

import (
```

```
    "context"
    "log"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

var (
    localPath string
    bucket    string
    prefix    string
)

func init() {
    if len(os.Args) != 4 {
        log.Fatalln("Usage:", os.Args[0], "<local path> <bucket> <prefix>")
    }
    localPath = os.Args[1]
    bucket = os.Args[2]
    prefix = os.Args[3]
}

func main() {
    walker := make(fileWalk)
    go func() {
        // Gather the files to upload by walking the path recursively
        if err := filepath.Walk(localPath, walker.Walk); err != nil {
            log.Fatalln("Walk failed:", err)
        }
        close(walker)
    }()

    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalln("error:", err)
    }

    // For each file found walking, upload it to Amazon S3
    uploader := manager.NewUploader(s3.NewFromConfig(cfg))
    for path := range walker {
```

```
        rel, err := filepath.Rel(localPath, path)
        if err != nil {
            log.Fatalln("Unable to get relative path:", path, err)
        }
        file, err := os.Open(path)
        if err != nil {
            log.Println("Failed opening file", path, err)
            continue
        }
        defer file.Close()
        result, err := uploader.Upload(context.TODO(), &s3.PutObjectInput{
            Bucket: &bucket,
            Key:    aws.String(filepath.Join(prefix, rel)),
            Body:   file,
        })
        if err != nil {
            log.Fatalln("Failed to upload", path, err)
        }
        log.Println("Uploaded", path, result.Location)
    }
}

type fileWalk chan string

func (f fileWalk) Walk(path string, info os.FileInfo, err error) error {
    if err != nil {
        return err
    }
    if !info.IsDir() {
        f <- path
    }
    return nil
}
```

Download Manager

The Amazon S3 Downloader manager determines if a file can be split into smaller parts and downloaded in parallel. You can customize the number of parallel downloads and the size of the downloaded parts.

Example: Download a File

The following example uses the Amazon S3 `Downloader` to download a file. Using `Downloader` is similar to the s3.GetObject operation.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/feature/s3/manager"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Println("error:", err)
    return
}

client := s3.NewFromConfig(cfg)

downloader := manager.NewDownloader(client)
numBytes, err := downloader.Download(context.TODO(), downloadFile, &s3.GetObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
})
```

The `downloadFile` parameter is an `io.WriterAt` type. The `WriterAt` interface enables the `Downloader` to write multiple parts of the file in parallel.

Configuration Options

When you instantiate a `Downloader` instance, you can specify configuration options to customize how objects are downloaded:

- `PartSize` – Specifies the buffer size, in bytes, of each part to download. The minimum size per part is 5 MB.
- `Concurrency` – Specifies the number of parts to download in parallel.

The `Concurrency` value limits the concurrent number of part download that can occur for a given `Download` call. This is not a global client concurrency limit. Tweak the `PartSize` and

`Concurrency` configuration values to find the optimal configuration. For example, systems with high-bandwidth connections can receive bigger parts and more downloads in parallel.

For example, your application configures `Downloader` with a `Concurrency` of `5`. Your application then calls `Download` from two different goroutines, the result will be `10` concurrent part downloads (2 goroutines * 5 `Concurrency`).

> **Warning**
>
> Your application is expected to limit the concurrent calls to `Download` to prevent application resource exhaustion.

For more information about `Downloader` and its other configuration options, see manager.Downloader in the AWS SDK for Go API Reference.

Overriding Downloader Options Per Download

You can override the `Downloader` options when calling `Download` by providing one or more functional arguments to the method. These overrides are concurrency safe modifications and do not affect ongoing uploads, or subsequent `Download` calls to the manager. For example, to override the `PartSize` configuration for a specific upload request:

```
params := &s3.GetObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("my-key"),
}
resp, err := downloader.Download(context.TODO(), targetWriter, params, func(u
 *manager.Downloader) {
    u.PartSize = 10 * 1024 * 1024, // 10 MiB
})
```

Examples

Download All Objects in a Bucket

The following example uses pagination to gather a list of objects from an Amazon S3 bucket. Then it downloads each object to a local file.

```
package main
```

```
import (
    "context"
    "fmt"
    "log"
    "os"
    "path/filepath"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

var (
    Bucket         = "amzn-s3-demo-bucket" // Download from this bucket
    Prefix         = "logs/"    // Using this key prefix
    LocalDirectory = "s3logs"   // Into this directory
)

func main() {
    cfg, err := config.LoadDefaultConfig(context.TODO())
    if err != nil {
        log.Fatalln("error:", err)
    }

    client := s3.NewFromConfig(cfg)
    manager := manager.NewDownloader(client)

    paginator := s3.NewListObjectsV2Paginator(client, &s3.ListObjectsV2Input{
        Bucket: &Bucket,
        Prefix: &Prefix,
    })

    for paginator.HasMorePages() {
        page, err := paginator.NextPage(context.TODO())
        if err != nil {
            log.Fatalln("error:", err)
        }
        for _, obj := range page.Contents {
            if err := downloadToFile(manager, LocalDirectory, Bucket,
 aws.ToString(obj.Key)); err != nil {
                log.Fatalln("error:", err)
            }
        }
```

```
    }
}

func downloadToFile(downloader *manager.Downloader, targetDirectory, bucket, key
 string) error {
    // Create the directories in the path
    file := filepath.Join(targetDirectory, key)
    if err := os.MkdirAll(filepath.Dir(file), 0775); err != nil {
        return err
    }

    // Set up the local file
    fd, err := os.Create(file)
    if err != nil {
        return err
    }
    defer fd.Close()

    // Download the file using the AWS SDK for Go
    fmt.Printf("Downloading s3://%s/%s to %s...\n", bucket, key, file)
    _, err = downloader.Download(context.TODO(), fd, &s3.GetObjectInput{Bucket:
 &bucket, Key: &key})

    return err
}
```

GetBucketRegion

GetBucketRegion is a utility function for determining the AWS Region location of an Amazon S3 Bucket. `GetBucketRegion` takes an Amazon S3 client and uses it to determine the location of the requested Bucket within the AWS Partition associated with the client's configured Region.

For example, to find the Region for the Bucket *amzn-s3-demo-bucket*:

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    log.Println("error:", err)
    return
}

bucket := "amzn-s3-demo-bucket"
region, err := manager.GetBucketRegion(ctx, s3.NewFromConfig(cfg), bucket)
if err != nil {
```

```
    var bnf manager.BucketNotFound
    if errors.As(err, &bnf) {
        log.Printf("unable to find bucket %s's Region\n", bucket)
    } else {
        log.Println("error:", err)
    }
    return
}
fmt.Printf("Bucket %s is in %s region\n", bucket, region)
```

If `GetBucketRegion` is not able to resolve the location of a Bucket, the function returns a BucketNotFound error type as shown in the example.

Unseekable Streaming Input

For API operations like `PutObject` and `UploadPart`, the Amazon S3 client expects the value of the `Body` input parameter to implement the io.Seeker interface by default. The `io.Seeker` interface is used by the client to determine the length of the value to upload, and to compute payload hash for the request signature. If the `Body` input parameter value does not implement `io.Seeker`, your application will receive an error.

```
operation error S3: PutObject, failed to compute payload hash: failed to seek
body to start, request stream is not seekable
```

You can change this behavior by modifying the operation method's Middleware using functional options. The WithAPIOptions helper returns a functional option for zero or more middleware mutators. To disable the client computing the payload hash and use Unsigned Payload request signature add v4.SwapComputePayloadSHA256ForUnsignedPayloadMiddleware.

```
resp, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
    Bucket: &bucketName,
    Key: &objectName,
    Body: bytes.NewBuffer([]byte(`example object!`)),
    ContentLength: 15, // length of body
}, s3.WithAPIOptions(
    v4.SwapComputePayloadSHA256ForUnsignedPayloadMiddleware,
))
```

Warning

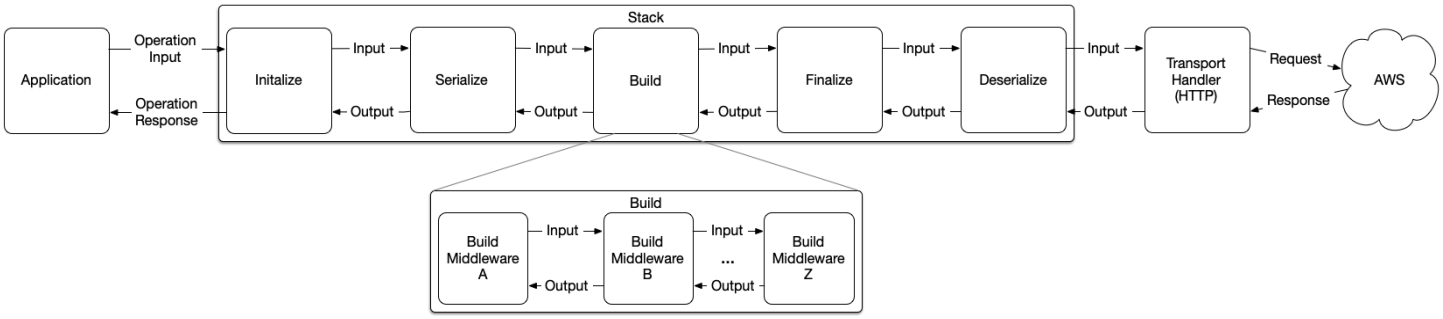
Amazon S3 requires the content length to be provided for all object's uploaded to a bucket. Since the `Body` input parameter does not implement `io.Seeker` interface, the client will not be able to compute the `ContentLength` parameter for the request. The parameter must be provided by the application. The request will fail if the `ContentLength` parameter is not provided.

Use the SDK's Amazon S3 Upload Manager for uploads that are not seekable and do not have a known length.

Customizing the AWS SDK for Go v2 Client Requests with Middleware

> **Warning**
>
> Modifying the client request pipeline can result in malformed/invalid requests, or can result in unexpected application errors. This functionality is meant for advanced uses cases not provided by the SDK interface by default.

You can customize AWS SDK for Go client requests by registering one or more middleware to a service operation's stack. The stack is composed of a series of steps: Initialize, Serialize, Build, Finalize, and Deserialize. Each step contains zero or more middleware that operate on that step's input and output types. The following diagram and table provide an overview of how an operation's request and response traverses the stack.



Stack Step	Description
Initialize	Prepares the input, and sets any default parameters as needed.
Serialize	Serializes the input to a protocol format suitable for the target transport layer.
Build	Attach additional metadata to the serialized input, such as HTTP Content-Length.

Stack Step	Description
Finalize	Final message preparation, including retries and authentication (SigV4 signing).
Deserialize	Deserialize responses from the protocol format into a structured type or error.

Each middleware within a given step must have a unique identifier, which is determined by the middleware's `ID` method. Middleware identifiers ensure that only one instance of a given middleware is registered to a step, and allows other step middleware to be inserted relative to it.

You attach step middleware by using a step's `Insert` or `Add` methods. You use `Add` to attach a middleware to the beginning of a step by specifying middleware.Before as the RelativePosition, and middleware.After to attach to the end of the step. You use `Insert` to attach a middleware to a step by inserting the middleware relative to another step middleware.

> **Warning**
>
> You must use the `Add` method to safely insert custom step middleware. Using `Insert` creates a dependency between your custom middleware, and the middleware that you are inserting relative to. The middleware within a stack step must be considered opaque to avoid breaking changes occurring to your application.

Writing a Custom Middleware

Each stack step has an interface that you must satisfy in order attach a middleware to a given step. You can use one of the provided *`Step`*`MiddlewareFunc` functions to quickly satisfy this interface. The following table outlines the steps, their interface, and the helper function that can be used to satisfy the interface.

Step	Interface	Helper Function
Initialize	InitializeMiddleware	InitializeMiddlewareFunc
Build	BuildMiddleware	BuildMiddlewareFunc

Step	Interface	Helper Function
Serialize	SerializeMiddleware	SerializeMiddlewareFunc
Finalize	FinalizeMiddleware	FinalizeMiddlewareFunc
Deserialize	DeserializeMiddleware	DeserializeMiddlewareFunc

The following examples show how you can write a custom middleware to populate the Bucket member of the Amazon S3 `GetObject` API calls if one is not provided. This middleware will be referenced in proceeding examples to show how to attach step middleware to the stack.

```
import "github.com/aws/smithy-go/aws"
import "github.com/aws/smithy-go/middleware"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

var defaultBucket = middleware.InitializeMiddlewareFunc("DefaultBucket", func(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    // Type switch to check if the input is s3.GetObjectInput, if so and the bucket is
 not set, populate it with
    // our default.
    switch v := in.Parameters.(type) {
    case *s3.GetObjectInput:
        if v.Bucket == nil {
            v.Bucket = aws.String("amzn-s3-demo-bucket")
        }
    }

    // Middleware must call the next middleware to be executed in order to continue
 execution of the stack.
    // If an error occurs, you can return to prevent further execution.
    return next.HandleInitialize(ctx, in)
})
```

Attaching Middleware to All Clients

You can attach your custom step middleware to every client by adding the middleware using the `APIOptions` member of the aws.Config type. The following examples attaches the `defaultBucket` middleware to every client constructed using your applications `aws.Config` object:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

cfg.APIOptions = append(cfg.APIOptions, func(stack *middleware.Stack) error {
    // Attach the custom middleware to the beginning of the Initialize step
    return stack.Initialize.Add(defaultBucket, middleware.Before)
})

client := s3.NewFromConfig(cfg)
```

Attaching Middleware to a Specific Operation

You can attach your custom step middleware to a specific client operation by modifying the client's `APIOptions` member using the variadic argument list for an operation. The following examples attaches the `defaultBucket` middleware to a specific Amazon S3 `GetObject` operation invocation:

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...
```

```
// registerDefaultBucketMiddleware registers the defaultBucket middleware with the
 provided stack.
func registerDefaultBucketMiddleware(stack *middleware.Stack) error {
    // Attach the custom middleware to the beginning of the Initialize step
    return stack.Initialize.Add(defaultBucket, middleware.Before)
}

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg)

object, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    Key: aws.String("my-key"),
}, func(options *s3.Options) {
    // Register the defaultBucketMiddleware for this operation only
    options.APIOptions = append(options.APIOptions, registerDefaultBucketMiddleware)
})
```

Passing Metadata Down the Stack

In certain situations, you may find that you require two or more middleware to function in tandem by sharing information or state. You can use context.Context to pass this metadata by using middleware.WithStackValue. `middleware.WithStackValue` attaches the given key-value pair to the provided context, and safely limits the scope to the currently executing stack. These stack-scoped values can be retrieved from a context using middleware.GetStackValue and providing the key used to stored the corresponding value. Keys must be comparable, and you must define your own types as context keys to avoid collisions. The following examples shows how two middleware can use `context.Context` to pass information down the stack.

```
import "context"
import "github.com/aws/smithy-go/middleware"

// ...

type customKey struct {}
```

```
func GetCustomKey(ctx context.Context) (v string) {
    v, _ = middleware.GetStackValue(ctx, customKey{}).(string)
    return v
}

func SetCustomKey(ctx context.Context, value string) context.Context {
    return middleware.WithStackValue(ctx, customKey{}, value)
}

// ...

var customInitalize = middleware.InitializeMiddlewareFunc("customInitialize", func(
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    ctx = SetCustomKey(ctx, "my-custom-value")

    return next.HandleInitialize(ctx, in)
})

var customBuild = middleware.BuildMiddlewareFunc("customBuild", func(
    ctx context.Context, in middleware.BuildInput, next middleware.BuildHandler,
) (
    out middleware.BuildOutput, metadata middleware.Metadata, err error,
) {
    customValue := GetCustomKey(ctx)

    // use customValue

    return next.HandleBuild(ctx, in)
})
```

Metadata Provided by the SDK

The AWS SDK for Go provides several metadata values that can be retrieved from the provided context. These values can be used to enable more dynamic middleware that modifies its behavior based on the executing service, operation, or target region. A few of the available keys are provided in the table below:

Key	Retriever	Description
ServiceID	GetServiceID	Retrieve the service identifier for the executing stack. This can be compared to the service client package's `ServiceID` constant.
OperationName	GetOperationName	Retrieve the operation name for the executing stack.
Logger	GetLogger	Retrieve the logger that can be used for logging message from the middleware.

Passing Metadata Up the Stack

You can pass metadata up through the stack by adding metadata key and value pairs using the middleware.Metadata. Each middleware step returns an output structure, metadata, and an error. Your custom middleware must return the metadata received from calling the next handler in the step. This ensures that metadata added by downstream middleware propagates to the application invoking the service operation. The resulting metadata is accessible to the invoking application by either the operation's output shape via the `ResultMetadata` structure member.

The following examples shows how a custom middleware can add metadata that is returned as part of the operation output.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/smithy-go/middleware"

// ...

type customKey struct{}

func GetCustomKey(metadata middleware.Metadata) (v string) {
    v, _ = metadata.Get(customKey{}).(string)
    return v
}
```

```
func SetCustomKey(metadata *middleware.Metadata, value string) {
    metadata.Set(customKey{}, value)
}

// ...

var customInitalize = middleware.InitializeMiddlewareFunc("customInitialize", func (
    ctx context.Context, in middleware.InitializeInput, next
 middleware.InitializeHandler,
) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleInitialize(ctx, in)
    if err != nil {
        return out, metadata, err
    }

    SetCustomKey(&metadata, "my-custom-value")

    return out, metadata, nil
})

// ...

client := s3.NewFromConfig(cfg, func (options *s3.Options) {
    options.APIOptions = append(options.APIOptions, func(stack *middleware.Stack) error
 {
        return stack.Initialize.Add(customInitalize, middleware.After)
    })
})

out, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    // handle error
}

customValue := GetCustomKey(out.ResponseMetadata)
```

Frequently Asked Questions

How do I configure my SDK's HTTP client? Are there any guidelines or best practices?

We are unable to provide guidance to customers on how to configure their HTTP workflow in a manner that is most effective for their particular workload. The answer to this is the product of a multivariate equation, with input factors including but not limited to:

- the network footprint of the application (TPS, throughput, etc.)
- the services being used
- the compute characteristics of the deployment
- the geographical nature of the deployment
- the desired application behavior or needs of the application itself (SLAs, timings, etc.)

How should I configure operation timeouts?

Much like the previous question, it depends. Elements to consider here include the following:

- All of the above factors concerning HTTP client config
- Your own application timing or SLA constraints (e.g. if you yourself serve traffic to other consumers)

The answer to this question should almost NEVER be based on pure empirical observation of upstream behavior - e.g. "I made 1000 calls to this operation, it took at most 5 seconds so I will set the timeout based on that with a safety factor of 2x to 10 seconds". Environment conditions can change, services can temporarily degrade, and these types of assumptions can become wrong without warning.

Requests made by the SDK are timing out or taking too long, how do I fix this?

We are unable to assist with extended or timed-out operation calls due to extended time spent on the wire. "Wire time" in the SDK is defined as any of the following:

- Time spent in an SDK client's `HTTPClient.Do()` method
- Time spent in `Read()`s on an HTTP response body that has been forwarded to the caller (e.g. `GetObject`)

If you are experiencing issues due to operation latency or timeouts, your first course of action should be to obtain telemetry of the SDK operation lifecycle to determine the timing breakdown between time spent on the wire and the surrounding overhead of the operation. See the guide on timing SDK operations, which contains a reusable code snippet that can achieve this.

How do I fix a `read: connection reset` error?

The SDK retries any errors matching the `connection reset` pattern by default. This will cover error handling for most operations, where the operation's HTTP response is fully consumed and deserialized into its modeled result type.

However, this error can still occur in a context **outside** of the retry loop: certain service operations directly forward the API's HTTP response body to the caller to be consumed from the wire directly via `io.ReadCloser` (e.g. `GetObject`'s object payload). You may encounter this error when performing a `Read` on the response body.

This error indicates that your host, the service or any intermediary party (e.g. NAT gateways, proxies, load balancers) closed the connection while attempting to read the response.

This can occur for several reasons:

- You did not consume the response body for some time after the response itself was received (after the service operation was called). **We recommend you consume the HTTP response body as soon as possible for these types of operations.**
- You did not close a previously-received response body. This can cause connection resets on certain platforms. **You MUST close any `io.ReadCloser` instances provided in an operation's response, regardless of whether you consume its contents.**

Beyond that, try running a `tcpdump` for an affected connection at the edge of your network (e.g. after any proxies that you control). If you see that the AWS endpoint seems to be sending a TCP RST, you should use the AWS support console to open a case against the offending service. Be prepared to provide request IDs and specific timestamps of when the issue occured.

Why am I getting "invalid signature" errors when using an HTTP proxy with the SDK?

The signature algorithm for AWS services (generally sigv4) is tied to the serialized request's headers, more specifically most headers prefixed with `X-`. Proxies are prone to modifying the outgoing request by adding additional forwarding information (often via an `X-Forwarded-For` header) which effectively breaks the signature that the SDK calculated.

If you're using an HTTP proxy and experiencing signature errors, you should work to capture the request **as it appears outgoing from the proxy** and determine whether it is different.

Timing SDK operations

When debugging timeout/latency issues in the SDK, it is critical to identify the components of the operation lifecycle which are taking more time to execute than expected. As a starting point, you will generally need to inspect the timing breakdown between the overall operation call and the HTTP call itself.

The following sample program implements a basic instrumentation probe in terms of `smithy-go` middleware for SQS clients and demonstrates how it is used. The probe emits the following information for each operation call:

- AWS request ID
- service ID
- operation name
- operation invocation time
- http call time

Each emitted message is prefixed with a unique (to a single operation) "invocation ID" which is set at the beginning of the handler stack.

The entry point for instrumentation is exposed as `WithOperationTiming`, which is parameterized to accept a message handling function which will receive instrumentation "events" as formatted strings. `PrintfMSGHandler` is provided as a convenience which will simply dump messages to stdout.

The service used here is interchangeable - ALL service client options accept `APIOptions` and an `HTTPClient` as configuration. For example, `WithOperationTiming` could instead be declared as:

```
func WithOperationTiming(msgHandler func(string)) func(*s3.Options)
func WithOperationTiming(msgHandler func(string)) func(*dynamodb.Options)
// etc.
```

If you change it, be sure to change the signature of the function it returns as well.

```
import (
    "context"
    "fmt"
    "log"
    "net/http"
    "sync"
    "time"

    awsmiddleware "github.com/aws/aws-sdk-go-v2/aws/middleware"
    awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/sqs"
    "github.com/aws/smithy-go/middleware"
    smithyrand "github.com/aws/smithy-go/rand"
)

// WithOperationTiming instruments an SQS client to dump timing information for
// the following spans:
//   - overall operation time
//   - HTTPClient call time
//
// This instrumentation will also emit the request ID, service name, and
// operation name for each invocation.
//
// Accepts a message "handler" which is invoked with formatted messages to be
// handled externally, you can use the declared PrintfMSGHandler to simply dump
// these values to stdout.
func WithOperationTiming(msgHandler func(string)) func(*sqs.Options) {
    return func(o *sqs.Options) {
        o.APIOptions = append(o.APIOptions, addTimingMiddlewares(msgHandler))
        o.HTTPClient = &timedHTTPClient{
            client:     awshttp.NewBuildableClient(),
            msgHandler: msgHandler,
        }
    }
}
```

```
// PrintfMSGHandler writes messages to stdout.
func PrintfMSGHandler(msg string) {
    fmt.Printf("%s\n", msg)
}

type invokeIDKey struct{}

func setInvokeID(ctx context.Context, id string) context.Context {
    return middleware.WithStackValue(ctx, invokeIDKey{}, id)
}

func getInvokeID(ctx context.Context) string {
    id, _ := middleware.GetStackValue(ctx, invokeIDKey{}).(string)
    return id
}

// Records the current time, and returns a function to be called when the
// target span of events is completed. The return function will emit the given
// span name and time elapsed to the given message consumer.
func timeSpan(ctx context.Context, name string, consumer func(string)) func() {
    start := time.Now()
    return func() {
        elapsed := time.Now().Sub(start)
        consumer(fmt.Sprintf("[%s] %s: %s", getInvokeID(ctx), name, elapsed))
    }
}

type timedHTTPClient struct {
    client     *awshttp.BuildableClient
    msgHandler func(string)
}

func (c *timedHTTPClient) Do(r *http.Request) (*http.Response, error) {
    defer timeSpan(r.Context(), "http", c.msgHandler)()

    resp, err := c.client.Do(r)
    if err != nil {
        return nil, fmt.Errorf("inner client do: %v", err)
    }

    return resp, nil
}

type addInvokeIDMiddleware struct {
```

```
    msgHandler func(string)
}

func (*addInvokeIDMiddleware) ID() string { return "addInvokeID" }

func (*addInvokeIDMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    out middleware.InitializeOutput, md middleware.Metadata, err error,
) {
    id, err := smithyrand.NewUUID(smithyrand.Reader).GetUUID()
    if err != nil {
        return out, md, fmt.Errorf("new uuid: %v", err)
    }

    return next.HandleInitialize(setInvokeID(ctx, id), in)
}

type timeOperationMiddleware struct {
    msgHandler func(string)
}

func (*timeOperationMiddleware) ID() string { return "timeOperation" }

func (m *timeOperationMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    middleware.InitializeOutput, middleware.Metadata, error,
) {
    defer timeSpan(ctx, "operation", m.msgHandler)()
    return next.HandleInitialize(ctx, in)
}

type emitMetadataMiddleware struct {
    msgHandler func(string)
}

func (*emitMetadataMiddleware) ID() string { return "emitMetadata" }

func (m *emitMetadataMiddleware) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    middleware.InitializeOutput, middleware.Metadata, error,
) {
    out, md, err := next.HandleInitialize(ctx, in)

    invokeID := getInvokeID(ctx)
```

```
    requestID, _ := awsmiddleware.GetRequestIDMetadata(md)
    service := awsmiddleware.GetServiceID(ctx)
    operation := awsmiddleware.GetOperationName(ctx)
    m.msgHandler(fmt.Sprintf(`[%s] requestID = "%s"`, invokeID, requestID))
    m.msgHandler(fmt.Sprintf(`[%s] service   = "%s"`, invokeID, service))
    m.msgHandler(fmt.Sprintf(`[%s] operation = "%s"`, invokeID, operation))

    return out, md, err
}

func addTimingMiddlewares(mh func(string)) func(*middleware.Stack) error {
    return func(s *middleware.Stack) error {
        if err := s.Initialize.Add(&timeOperationMiddleware{msgHandler: mh},
 middleware.Before); err != nil {
            return fmt.Errorf("add time operation middleware: %v", err)
        }
        if err := s.Initialize.Add(&addInvokeIDMiddleware{msgHandler: mh},
 middleware.Before); err != nil {
            return fmt.Errorf("add invoke id middleware: %v", err)
        }
        if err := s.Initialize.Insert(&emitMetadataMiddleware{msgHandler: mh},
 "RegisterServiceMetadata", middleware.After); err != nil {
            return fmt.Errorf("add emit metadata middleware: %v", err)
        }
        return nil
    }
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        log.Fatal(fmt.Errorf("load default config: %v", err))
    }

    svc := sqs.NewFromConfig(cfg, WithOperationTiming(PrintfMSGHandler))

    var wg sync.WaitGroup

    for i := 0; i < 6; i++ {
        wg.Add(1)
        go func() {
            defer wg.Done()

            _, err = svc.ListQueues(context.Background(), nil)
```

```
            if err != nil {
                fmt.Println(fmt.Errorf("list queues: %v", err))
            }
        }()
    }
    wg.Wait()
}
```

A sample output of this program:

```
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] http: 192.24067ms
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] requestID = "dbee3082-96a3-5b23-
adca-6d005696fa94"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] service   = "SQS"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] operation = "ListQueues"
[e9a801bb-c51d-45c8-8e9f-a202e263fde8] operation: 193.098393ms
[0740f0e0-953e-4328-94fc-830a5052e763] http: 195.185732ms
[0740f0e0-953e-4328-94fc-830a5052e763] requestID = "48b301fa-
fc9f-5f1f-9007-5c783caa9322"
[0740f0e0-953e-4328-94fc-830a5052e763] service   = "SQS"
[0740f0e0-953e-4328-94fc-830a5052e763] operation = "ListQueues"
[0740f0e0-953e-4328-94fc-830a5052e763] operation: 195.725491ms
[c0589832-f351-4cc7-84f1-c656eb79dbd7] http: 200.52383ms
[444030d0-6743-4de5-bd91-bc40b2b94c55] http: 200.525919ms
[c0589832-f351-4cc7-84f1-c656eb79dbd7] requestID = "4a73cc82-b47b-56e1-
b327-9100744e1b1f"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] service   = "SQS"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] operation = "ListQueues"
[c0589832-f351-4cc7-84f1-c656eb79dbd7] operation: 201.214365ms
[444030d0-6743-4de5-bd91-bc40b2b94c55] requestID = "ca1523ed-1879-5610-
bf5d-7e6fd84cabee"
[444030d0-6743-4de5-bd91-bc40b2b94c55] service   = "SQS"
[444030d0-6743-4de5-bd91-bc40b2b94c55] operation = "ListQueues"
[444030d0-6743-4de5-bd91-bc40b2b94c55] operation: 201.197071ms
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] http: 206.449568ms
[12b2b39d-df86-4648-a436-ff0482d13340] http: 206.526603ms
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] requestID = "64229710-b552-56ed-8f96-
ca927567ec7b"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] service   = "SQS"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] operation = "ListQueues"
[079e8dbd-bb93-43ab-89e5-a7bb392b86a5] operation: 207.252357ms
[12b2b39d-df86-4648-a436-ff0482d13340] requestID =
 "76d9cbc0-07aa-58aa-98b7-9642c79f9851"
```

```
[12b2b39d-df86-4648-a436-ff0482d13340] service   = "SQS"
[12b2b39d-df86-4648-a436-ff0482d13340] operation = "ListQueues"
[12b2b39d-df86-4648-a436-ff0482d13340] operation: 207.360621ms
```

Unit Testing with the AWS SDK for Go v2

When using the SDK in your application, you'll want to mock out the SDK for your application's unit test. Mocking out the SDK allows your test to be focused on what you want to test, not the internals of the SDK.

To support mocking, use Go interfaces instead of concrete service client, paginators, and waiter types, such as `s3.Client`. This allows your application to use patterns like dependency injection to test your application logic.

Mocking Client Operations

In this example, `S3GetObjectAPI` is an interface that defines the set of Amazon S3 API operations required by the `GetObjectFromS3` function. `S3GetObjectAPI` is satisfied by the Amazon S3 client's GetObject method.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type S3GetObjectAPI interface {
    GetObject(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error)
}

func GetObjectFromS3(ctx context.Context, api S3GetObjectAPI, bucket, key string)
 ([]byte, error) {
    object, err := api.GetObject(ctx, &s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return ioutil.ReadAll(object.Body)
}
```

To test the `GetObjectFromS3` function, use the `mockGetObjectAPI` to satisfy the `S3GetObjectAPI` interface definition. Then use the `mockGetObjectAPI` type to mock output and error responses returned from the service client.

```
import "testing"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type mockGetObjectAPI func(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error)

func (m mockGetObjectAPI) GetObject(ctx context.Context, params *s3.GetObjectInput,
 optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error) {
    return m(ctx, params, optFns...)
}

func TestGetObjectFromS3(t *testing.T) {
    cases := []struct {
        client func(t *testing.T) S3GetObjectAPI
        bucket string
        key string
        expect []byte
    }{
        {
            client: func(t *testing.T) S3GetObjectAPI {
                return mockGetObjectAPI(func(ctx context.Context, params
 *s3.GetObjectInput, optFns ...func(*s3.Options)) (*s3.GetObjectOutput, error) {
                    t.Helper()
                    if params.Bucket == nil {
                        t.Fatal("expect bucket to not be nil")
                    }
                    if e, a := "fooBucket", *params.Bucket; e != a {
                        t.Errorf("expect %v, got %v", e, a)
                    }
                    if params.Key == nil {
                        t.Fatal("expect key to not be nil")
                    }
                    if e, a := "barKey", *params.Key; e != a {
                        t.Errorf("expect %v, got %v", e, a)
                    }

                    return &s3.GetObjectOutput{
```

```
                        Body: ioutil.NopCloser(bytes.NewReader([]byte("this is the body
 foo bar baz"))),
                    }, nil
                })
            },
            bucket: "amzn-s3-demo-bucket>",
            key:    "barKey",
            expect: []byte("this is the body foo bar baz"),
        },
    }

    for i, tt := range cases {
        t.Run(strconv.Itoa(i), func(t *testing.T) {
            ctx := context.TODO()
            content, err := GetObjectFromS3(ctx, tt.client(t), tt.bucket, tt.key)
            if err != nil {
                t.Fatalf("expect no error, got %v", err)
            }
            if e, a := tt.expect, content; bytes.Compare(e, a) != 0 {
                t.Errorf("expect %v, got %v", e, a)
            }
        })
    }
}
```

Mocking Paginators

Similar to service clients, paginators can be mocked by defining a Go interface for the paginator. That interface would be used by your application's code. This allows the SDK's implementation to be used when your application is running, and a mocked implementation for testing.

In the following example, `ListObjectsV2Pager` is an interface that defines the behaviors for the Amazon S3 ListObjectsV2Paginator required by `CountObjects` function.

```
import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type ListObjectsV2Pager interface {
    HasMorePages() bool
    NextPage(context.Context, ...func(*s3.Options)) (*s3.ListObjectsV2Output, error)
```

```
}

func CountObjects(ctx context.Context, pager ListObjectsV2Pager) (count int, err error)
 {
    for pager.HasMorePages() {
        var output *s3.ListObjectsV2Output
        output, err = pager.NextPage(ctx)
        if err != nil {
            return count, err
        }
        count += int(output.KeyCount)
    }
    return count, nil
}
```

To test `CountObjects`, create the `mockListObjectsV2Pager` type to satisfy the `ListObjectsV2Pager` interface definition. Then use `mockListObjectsV2Pager` to replicate the paging behavior of output and error responses from the service operation paginator.

```
import "context"
import  "fmt"
import "testing"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

type mockListObjectsV2Pager struct {
    PageNum int
    Pages   []*s3.ListObjectsV2Output
}

func (m *mockListObjectsV2Pager) HasMorePages() bool {
    return m.PageNum < len(m.Pages)
}

func (m *mockListObjectsV2Pager) NextPage(ctx context.Context, f ...func(*s3.Options))
 (output *s3.ListObjectsV2Output, err error) {
    if m.PageNum >= len(m.Pages) {
        return nil, fmt.Errorf("no more pages")
    }
    output = m.Pages[m.PageNum]
    m.PageNum++
    return output, nil
```

```
}

func TestCountObjects(t *testing.T) {
    pager := &mockListObjectsV2Pager{
        Pages: []*s3.ListObjectsV2Output{
            {
                KeyCount: 5,
            },
            {
                KeyCount: 10,
            },
            {
                KeyCount: 15,
            },
        },
    }
    objects, err := CountObjects(context.TODO(), pager)
    if err != nil {
        t.Fatalf("expect no error, got %v", err)
    }
    if expect, actual := 30, objects; expect != actual {
        t.Errorf("expect %v, got %v", expect, actual)
    }
}
```

SDK for Go V2 code examples

The code examples in this topic show you how to use the AWS SDK for Go V2 with AWS.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Some services contain additional example categories that show how to leverage libraries or functions specific to the service.

Services

- API Gateway examples using SDK for Go V2
- Aurora examples using SDK for Go V2
- Amazon Bedrock examples using SDK for Go V2
- Amazon Bedrock Runtime examples using SDK for Go V2
- CloudFormation examples using SDK for Go V2
- CloudWatch Logs examples using SDK for Go V2
- Amazon Cognito Identity Provider examples using SDK for Go V2
- Amazon DocumentDB examples using SDK for Go V2
- DynamoDB examples using SDK for Go V2
- IAM examples using SDK for Go V2
- Kinesis examples using SDK for Go V2
- Lambda examples using SDK for Go V2
- Amazon MSK examples using SDK for Go V2
- Partner Central examples using SDK for Go V2
- Amazon RDS examples using SDK for Go V2
- Amazon Redshift examples using SDK for Go V2

- Amazon S3 examples using SDK for Go V2
- Amazon SNS examples using SDK for Go V2
- Amazon SQS examples using SDK for Go V2

API Gateway examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with API Gateway.

AWS community contributions are examples that were created and are maintained by multiple teams across AWS. To provide feedback, use the mechanism provided in the linked repositories.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- AWS community contributions

AWS community contributions

Build and test a serverless application

The following code example shows how to build and test a serverless application using API Gateway with Lambda and DynamoDB

SDK for Go V2

Shows how to build and test a serverless application that consists of an API Gateway with Lambda and DynamoDB using the Go SDK.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- API Gateway
- DynamoDB

- Lambda

Aurora examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Aurora.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions

Get started

Hello Aurora

The following code example shows how to get started using Aurora.

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"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Aurora client and list up to
 20
// DB clusters 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
 }
 auroraClient := rds.NewFromConfig(sdkConfig)
 const maxClusters = 20
 fmt.Printf("Let's list up to %v DB clusters.\n", maxClusters)
 output, err := auroraClient.DescribeDBClusters(
  ctx, &rds.DescribeDBClustersInput{MaxRecords: aws.Int32(maxClusters)})
 if err != nil {
  fmt.Printf("Couldn't list DB clusters: %v\n", err)
  return
 }
 if len(output.DBClusters) == 0 {
  fmt.Println("No DB clusters found.")
 } else {
  for _, cluster := range output.DBClusters {
   fmt.Printf("DB cluster %v has database %v.\n", *cluster.DBClusterIdentifier,
    *cluster.DatabaseName)
  }
 }
}
```

- For API details, see DescribeDBClusters in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to:

- Create a custom Aurora DB cluster parameter group and set parameter values.
- Create a DB cluster that uses the parameter group.
- Create a DB instance that contains a database.
- Take a snapshot of the DB cluster, then clean up resources.

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 (
 "aurora/actions"
 "context"
 "fmt"
 "log"
 "slices"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/google/uuid"
)
```

```
// GetStartedClusters is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Aurora to do the following:
//
// 1. Create a custom DB cluster parameter group and set parameter values.
// 2. Create an Aurora DB cluster that is configured to use the parameter group.
// 3. Create a DB instance in the DB cluster that contains a database.
// 4. Take a snapshot of the DB cluster.
// 5. Delete the DB instance, DB cluster, and parameter group.
type GetStartedClusters struct {
 sdkConfig  aws.Config
 dbClusters actions.DbClusters
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedClusters constructs a GetStartedClusters instance from a
 configuration.
// It uses the specified config to get an Amazon Relational Database Service (Amazon
 RDS)
// client and create wrappers for the actions used in the scenario.
func NewGetStartedClusters(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) GetStartedClusters {
 auroraClient := rds.NewFromConfig(sdkConfig)
 return GetStartedClusters{
  sdkConfig:  sdkConfig,
  dbClusters: actions.DbClusters{AuroraClient: auroraClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedClusters) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 clusterName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
```

```
 log.Println("Welcome to the Amazon Aurora DB Cluster demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine, parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 cluster := scenario.CreateCluster(ctx, clusterName, dbEngine, dbName,
 parameterGroup)
 scenario.helper.Pause(5)
 dbInstance := scenario.CreateInstance(ctx, cluster)
 scenario.DisplayConnection(cluster)
 scenario.CreateSnapshot(ctx, clusterName)
 scenario.Cleanup(ctx, dbInstance, cluster, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a specified
// database engine and create a DB cluster parameter group that is compatible with a
// selected engine family.
func (scenario GetStartedClusters) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBClusterParameterGroup {

 log.Printf("Checking for an existing DB cluster parameter group named %v.\n",
  parameterGroupName)
 parameterGroup, err := scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
 if err != nil {
  panic(err)
 }
 if parameterGroup == nil {
  log.Printf("Getting available database engine versions for %v.\n", dbEngine)
  engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine, "")
  if err != nil {
   panic(err)
  }

  familySet := map[string]struct{}{}
  for _, family := range engineVersions {
   familySet[*family.DBParameterGroupFamily] = struct{}{}
  }
  var families []string
```

```
  for family := range familySet {
   families = append(families, family)
  }
  sort.Strings(families)
  familyIndex := scenario.questioner.AskChoice("Which family do you want to use?\n",
 families)
  log.Println("Creating a DB cluster parameter group.")
  _, err = scenario.dbClusters.CreateParameterGroup(
   ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
  if err != nil {
   panic(err)
  }
  parameterGroup, err = scenario.dbClusters.GetParameterGroup(ctx,
 parameterGroupName)
  if err != nil {
   panic(err)
  }
 }
 log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tName: %v\n", *parameterGroup.DBClusterParameterGroupName)
 log.Printf("\tARN: %v\n", *parameterGroup.DBClusterParameterGroupArn)
 log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
 log.Printf("\tDescription: %v\n", *parameterGroup.Description)
 log.Println(strings.Repeat("-", 88))
 return parameterGroup

}

// SetUserParameters shows how to get the parameters contained in a custom parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedClusters) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
```

```
    lower, _ := strconv.Atoi(rangeSplit[0])
    upper, _ := strconv.Atoi(rangeSplit[1])
    newValue := scenario.questioner.AskInt(
     fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
     demotools.InIntRange{Lower: lower, Upper: upper})
    dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
    updateParams = append(updateParams, dbParam)
   }
  }
  err = scenario.dbClusters.UpdateParameters(ctx, parameterGroupName, updateParams)
  if err != nil {
   panic(err)
  }
  log.Println("You can get a list of parameters you've set by specifying a source of
  'user'.")
  userParameters, err := scenario.dbClusters.GetParameters(ctx, parameterGroupName,
  "user")
  if err != nil {
   panic(err)
  }
  log.Println("Here are the parameters you've set:")
  for _, param := range userParameters {
   log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
  }
  log.Println(strings.Repeat("-", 88))
}

// CreateCluster shows how to create an Aurora DB cluster that contains a database
// of a specified type. The database is also configured to use a custom DB cluster
// parameter group.
func (scenario GetStartedClusters) CreateCluster(ctx context.Context, clusterName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBClusterParameterGroup) *types.DBCluster {

 log.Println("Checking for an existing DB cluster.")
 cluster, err := scenario.dbClusters.GetDbCluster(ctx, clusterName)
 if err != nil {
  panic(err)
 }
 if cluster == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator user name for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.Ask(
```

```
    "Enter a password for the administrator (at least 8 characters): ",
  demotools.NotEmpty{})
   engineVersions, err := scenario.dbClusters.GetEngineVersions(ctx, dbEngine,
  *parameterGroup.DBParameterGroupFamily)
   if err != nil {
    panic(err)
   }
   var engineChoices []string
   for _, engine := range engineVersions {
    engineChoices = append(engineChoices, *engine.EngineVersion)
   }
   log.Println("The available engines for your parameter group are:")
   engineIndex := scenario.questioner.AskChoice("Which engine do you want to use?\n",
  engineChoices)
   log.Printf("Creating DB cluster %v and database %v.\n", clusterName, dbName)
   log.Printf("The DB cluster is configured to use\nyour custom parameter group %v
 \n",
    *parameterGroup.DBClusterParameterGroupName)
   log.Printf("and selected engine %v.\n", engineChoices[engineIndex])
   log.Println("This typically takes several minutes.")
   cluster, err = scenario.dbClusters.CreateDbCluster(
    ctx, clusterName, *parameterGroup.DBClusterParameterGroupName, dbName, dbEngine,
    engineChoices[engineIndex], adminUsername, adminPassword)
   if err != nil {
    panic(err)
   }
   for *cluster.Status != "available" {
    scenario.helper.Pause(30)
    cluster, err = scenario.dbClusters.GetDbCluster(ctx, clusterName)
    if err != nil {
     panic(err)
    }
    log.Println("Cluster created and available.")
   }
  }
  log.Println("Cluster data:")
  log.Printf("\tDBClusterIdentifier: %v\n", *cluster.DBClusterIdentifier)
  log.Printf("\tARN: %v\n", *cluster.DBClusterArn)
  log.Printf("\tStatus: %v\n", *cluster.Status)
  log.Printf("\tEngine: %v\n", *cluster.Engine)
  log.Printf("\tEngine version: %v\n", *cluster.EngineVersion)
  log.Printf("\tDBClusterParameterGroup: %v\n", *cluster.DBClusterParameterGroup)
  log.Printf("\tEngineMode: %v\n", *cluster.EngineMode)
  log.Println(strings.Repeat("-", 88))
```

```
 return cluster
}

// CreateInstance shows how to create a DB instance in an existing Aurora DB
 cluster.
// A new DB cluster contains no DB instances, so you must add one. The first DB
 instance
// that is added to a DB cluster defaults to a read-write DB instance.
func (scenario GetStartedClusters) CreateInstance(ctx context.Context, cluster
 *types.DBCluster) *types.DBInstance {
 log.Println("Checking for an existing database instance.")
 dbInstance, err := scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
 if err != nil {
  panic(err)
 }
 if dbInstance == nil {
  log.Println("Let's create a database instance in your DB cluster.")
  log.Println("First, choose a DB instance type:")
  instOpts, err := scenario.dbClusters.GetOrderableInstances(
   ctx, *cluster.Engine, *cluster.EngineVersion)
  if err != nil {
   panic(err)
  }
  var instChoices []string
  for _, opt := range instOpts {
   instChoices = append(instChoices, *opt.DBInstanceClass)
  }
  slices.Sort(instChoices)
  instChoices = slices.Compact(instChoices)
  instIndex := scenario.questioner.AskChoice(
   "Which DB instance class do you want to use?\n", instChoices)
  log.Println("Creating a database instance. This typically takes several minutes.")
  dbInstance, err = scenario.dbClusters.CreateInstanceInCluster(
   ctx, *cluster.DBClusterIdentifier, *cluster.DBClusterIdentifier, *cluster.Engine,
   instChoices[instIndex])
  if err != nil {
   panic(err)
  }
  for *dbInstance.DBInstanceStatus != "available" {
   scenario.helper.Pause(30)
   dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *cluster.DBClusterIdentifier)
   if err != nil {
```

```
    panic(err)
   }
  }
 }
 log.Println("Instance data:")
 log.Printf("\tDBInstanceIdentifier: %v\n", *dbInstance.DBInstanceIdentifier)
 log.Printf("\tARN: %v\n", *dbInstance.DBInstanceArn)
 log.Printf("\tStatus: %v\n", *dbInstance.DBInstanceStatus)
 log.Printf("\tEngine: %v\n", *dbInstance.Engine)
 log.Printf("\tEngine version: %v\n", *dbInstance.EngineVersion)
 log.Println(strings.Repeat("-", 88))
 return dbInstance
}

// DisplayConnection displays connection information about an Aurora DB cluster and
 tips
// on how to connect to it.
func (scenario GetStartedClusters) DisplayConnection(cluster *types.DBCluster) {
 log.Println(
  "You can now connect to your database using your favorite MySql client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n" +
   "that is running in the same VPC as your database cluster. Pass the endpoint,\n"
 +
   "port, and administrator user name to 'mysql' and enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *cluster.Endpoint, *cluster.Port, *cluster.MasterUsername)
 log.Println("For more information, see the User Guide for Aurora:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/AuroraUserGuide/
CHAP_GettingStartedAurora.CreatingConnecting.Aurora.html#CHAP_GettingStartedAurora.Aurora.Co
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB cluster snapshot and wait until it's
 available.
func (scenario GetStartedClusters) CreateSnapshot(ctx context.Context, clusterName
 string) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB cluster (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", clusterName, scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.\n",
 snapshotId)
  snapshot, err := scenario.dbClusters.CreateClusterSnapshot(ctx, clusterName,
 snapshotId)
```

```
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.dbClusters.GetClusterSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBClusterSnapshotIdentifier: %v\n",
 *snapshot.DBClusterSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBClusterSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBClusterIdentifier: %v\n", *snapshot.DBClusterIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance, DB cluster, and DB cluster parameter
 group.
// Before the DB cluster parameter group can be deleted, all associated DB instances
 and
// DB clusters must first be deleted.
func (scenario GetStartedClusters) Cleanup(ctx context.Context, dbInstance
 *types.DBInstance, cluster *types.DBCluster,
 parameterGroup *types.DBClusterParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance, DB cluster, and parameter group
 (y/n)? ", "y") {
  log.Printf("Deleting database instance %v.\n", *dbInstance.DBInstanceIdentifier)
  err := scenario.dbClusters.DeleteInstance(ctx, *dbInstance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting database cluster %v.\n", *cluster.DBClusterIdentifier)
  err = scenario.dbClusters.DeleteDbCluster(ctx, *cluster.DBClusterIdentifier)
  if err != nil {
   panic(err)
```

```
  }
  log.Println(
   "Waiting for the DB instance and DB cluster to delete. This typically takes
 several minutes.")
  for dbInstance != nil || cluster != nil {
   scenario.helper.Pause(30)
   if dbInstance != nil {
    dbInstance, err = scenario.dbClusters.GetInstance(ctx,
 *dbInstance.DBInstanceIdentifier)
    if err != nil {
     panic(err)
    }
   }
   if cluster != nil {
    cluster, err = scenario.dbClusters.GetDbCluster(ctx,
 *cluster.DBClusterIdentifier)
    if err != nil {
     panic(err)
    }
   }
  }
  log.Printf("Deleting parameter group %v.",
 *parameterGroup.DBClusterParameterGroupName)
  err = scenario.dbClusters.DeleteParameterGroup(ctx,
 *parameterGroup.DBClusterParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

```
// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Define functions that are called by the scenario to manage Aurora actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}


// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
```

```
  return &output.DBClusterParameterGroups[0], err
 }
}


// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context, parameterGroupName
 string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
```

```
 }
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified engine
 and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
```

```
  DatabaseName:                aws.String(dbName),
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context, clusterName
 string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
```

```
 } else {
  return output.DBClusterSnapshot, nil
 }
}



// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context, clusterName
 string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

```
// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

```
// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
```

```
  }
 }
 return instances, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateDBCluster
  - CreateDBClusterParameterGroup
  - CreateDBClusterSnapshot
  - CreateDBInstance
  - DeleteDBCluster
  - DeleteDBClusterParameterGroup
  - DeleteDBInstance
  - DescribeDBClusterParameterGroups
  - DescribeDBClusterParameters
  - DescribeDBClusterSnapshots
  - DescribeDBClusters
  - DescribeDBEngineVersions
  - DescribeDBInstances
  - DescribeOrderableDBInstanceOptions
  - ModifyDBClusterParameterGroup

Actions

`CreateDBCluster`

The following code example shows how to use `CreateDBCluster`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateDbCluster creates a DB cluster that is configured to use the specified
 parameter group.
// The newly created DB cluster contains a database that uses the specified engine
 and
// engine version.
func (clusters *DbClusters) CreateDbCluster(ctx context.Context, clusterName string,
 parameterGroupName string,
 dbName string, dbEngine string, dbEngineVersion string, adminName string,
 adminPassword string) (
 *types.DBCluster, error) {

 output, err := clusters.AuroraClient.CreateDBCluster(ctx,
 &rds.CreateDBClusterInput{
  DBClusterIdentifier:         aws.String(clusterName),
  Engine:                      aws.String(dbEngine),
  DBClusterParameterGroupName: aws.String(parameterGroupName),
  DatabaseName:                aws.String(dbName),
```

```
  EngineVersion:               aws.String(dbEngineVersion),
  MasterUserPassword:          aws.String(adminPassword),
  MasterUsername:              aws.String(adminName),
 })
 if err != nil {
  log.Printf("Couldn't create DB cluster %v: %v\n", clusterName, err)
  return nil, err
 } else {
  return output.DBCluster, err
 }
}
```

- For API details, see CreateDBCluster in *AWS SDK for Go API Reference*.

`CreateDBClusterParameterGroup`

The following code example shows how to use `CreateDBClusterParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// CreateParameterGroup creates a DB cluster parameter group that is based on the
 specified
// parameter group family.
func (clusters *DbClusters) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBClusterParameterGroup, error) {

 output, err := clusters.AuroraClient.CreateDBClusterParameterGroup(ctx,
  &rds.CreateDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   DBParameterGroupFamily:      aws.String(parameterGroupFamily),
   Description:                 aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBClusterParameterGroup, err
 }
}
```

- For API details, see CreateDBClusterParameterGroup in *AWS SDK for Go API Reference*.

`CreateDBClusterSnapshot`

The following code example shows how to use `CreateDBClusterSnapshot`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateClusterSnapshot creates a snapshot of a DB cluster.
func (clusters *DbClusters) CreateClusterSnapshot(ctx context.Context, clusterName
 string, snapshotName string) (
 *types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.CreateDBClusterSnapshot(ctx,
 &rds.CreateDBClusterSnapshotInput{
  DBClusterIdentifier:         aws.String(clusterName),
  DBClusterSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBClusterSnapshot, nil
 }
}
```

- For API details, see CreateDBClusterSnapshot in *AWS SDK for Go API Reference*.

`CreateDBInstance`

The following code example shows how to use `CreateDBInstance`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// CreateInstanceInCluster creates a database instance in an existing DB cluster.
 The first database that is
// created defaults to a read-write DB instance.
func (clusters *DbClusters) CreateInstanceInCluster(ctx context.Context, clusterName
 string, instanceName string,
 dbEngine string, dbInstanceClass string) (*types.DBInstance, error) {
 output, err := clusters.AuroraClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBClusterIdentifier:  aws.String(clusterName),
  Engine:               aws.String(dbEngine),
  DBInstanceClass:      aws.String(dbInstanceClass),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
```

```
  return output.DBInstance, nil
 }
}
```

- For API details, see CreateDBInstance in *AWS SDK for Go API Reference*.

`DeleteDBCluster`

The following code example shows how to use `DeleteDBCluster`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteDbCluster deletes a DB cluster without keeping a final snapshot.
func (clusters *DbClusters) DeleteDbCluster(ctx context.Context, clusterName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBCluster(ctx, &rds.DeleteDBClusterInput{
  DBClusterIdentifier: aws.String(clusterName),
```

```
  SkipFinalSnapshot:   aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete DB cluster %v: %v\n", clusterName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBCluster in *AWS SDK for Go API Reference*.

`DeleteDBClusterParameterGroup`

The following code example shows how to use `DeleteDBClusterParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// DeleteParameterGroup deletes the named DB cluster parameter group.
func (clusters *DbClusters) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := clusters.AuroraClient.DeleteDBClusterParameterGroup(ctx,
  &rds.DeleteDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBClusterParameterGroup in *AWS SDK for Go API Reference*.

`DeleteDBInstance`

The following code example shows how to use `DeleteDBInstance`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// DeleteInstance deletes a DB instance.
func (clusters *DbClusters) DeleteInstance(ctx context.Context, instanceName string)
 error {
 _, err := clusters.AuroraClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Go API Reference*.

`DescribeDBClusterParameterGroups`

The following code example shows how to use `DescribeDBClusterParameterGroups`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameterGroup gets a DB cluster parameter group by name.
func (clusters *DbClusters) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBClusterParameterGroup, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterParameterGroups(
  ctx, &rds.DescribeDBClusterParameterGroupsInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBClusterParameterGroups[0], err
 }
}
```

- For API details, see DescribeDBClusterParameterGroups in *AWS SDK for Go API Reference*.

`DescribeDBClusterParameters`

The following code example shows how to use `DescribeDBClusterParameters`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB cluster parameter
 group.
func (clusters *DbClusters) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBClusterParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator :=
 rds.NewDescribeDBClusterParametersPaginator(clusters.AuroraClient,
  &rds.DescribeDBClusterParametersInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Source:                      aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
```

```
  if err != nil {
   log.Printf("Couldn't get paramaeters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- For API details, see DescribeDBClusterParameters in *AWS SDK for Go API Reference*.

`DescribeDBClusterSnapshots`

The following code example shows how to use `DescribeDBClusterSnapshots`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}
```

```
// GetClusterSnapshot gets a DB cluster snapshot.
func (clusters *DbClusters) GetClusterSnapshot(ctx context.Context, snapshotName
 string) (*types.DBClusterSnapshot, error) {
 output, err := clusters.AuroraClient.DescribeDBClusterSnapshots(ctx,
  &rds.DescribeDBClusterSnapshotsInput{
   DBClusterSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBClusterSnapshots[0], nil
 }
}
```

- For API details, see DescribeDBClusterSnapshots in *AWS SDK for Go API Reference*.

`DescribeDBClusters`

The following code example shows how to use `DescribeDBClusters`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbClusters struct {
 AuroraClient *rds.Client
}



// GetDbCluster gets data about an Aurora DB cluster.
func (clusters *DbClusters) GetDbCluster(ctx context.Context, clusterName string)
 (*types.DBCluster, error) {
 output, err := clusters.AuroraClient.DescribeDBClusters(ctx,
  &rds.DescribeDBClustersInput{
   DBClusterIdentifier: aws.String(clusterName),
  })
 if err != nil {
  var notFoundError *types.DBClusterNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB cluster %v does not exist.\n", clusterName)
   err = nil
  } else {
   log.Printf("Couldn't get DB cluster %v: %v\n", clusterName, err)
  }
  return nil, err
 } else {
  return &output.DBClusters[0], err
 }
}
```

- For API details, see DescribeDBClusters in *AWS SDK for Go API Reference*.

`DescribeDBEngineVersions`

The following code example shows how to use `DescribeDBEngineVersions`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (clusters *DbClusters) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := clusters.AuroraClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Go API Reference*.

`DescribeDBInstances`

The following code example shows how to use `DescribeDBInstances`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (clusters *DbClusters) GetInstance(ctx context.Context, instanceName string) (
 *types.DBInstance, error) {
 output, err := clusters.AuroraClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
```

```
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Go API Reference*.

`DescribeOrderableDBInstanceOptions`

The following code example shows how to use `DescribeOrderableDBInstanceOptions`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
```

```
// compatible with a set of specifications.
func (clusters *DbClusters) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instances []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(clusters.AuroraClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instances: %v\n", err)
   break
  } else {
   instances = append(instances, output.OrderableDBInstanceOptions...)
  }
 }
 return instances, err
}
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for Go API Reference*.

`ModifyDBClusterParameterGroup`

The following code example shows how to use `ModifyDBClusterParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbClusters struct {
 AuroraClient *rds.Client
}



// UpdateParameters updates parameters in a named DB cluster parameter group.
func (clusters *DbClusters) UpdateParameters(ctx context.Context, parameterGroupName
 string, params []types.Parameter) error {
 _, err := clusters.AuroraClient.ModifyDBClusterParameterGroup(ctx,
  &rds.ModifyDBClusterParameterGroupInput{
   DBClusterParameterGroupName: aws.String(parameterGroupName),
   Parameters:                  params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see ModifyDBClusterParameterGroup in *AWS SDK for Go API Reference*.

Amazon Bedrock examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon Bedrock.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Actions

Get started

Hello Amazon Bedrock

The following code example shows how to get started using Amazon Bedrock.

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"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/bedrock"
)

const region = "us-east-1"

// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock client and
// list the available foundation models in your account and the chosen region.
// This example uses the default settings specified in your shared credentials
```

```
// and config files.
func main() {
 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx, config.WithRegion(region))
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }
 bedrockClient := bedrock.NewFromConfig(sdkConfig)
 result, err := bedrockClient.ListFoundationModels(ctx,
 &bedrock.ListFoundationModelsInput{})
 if err != nil {
  fmt.Printf("Couldn't list foundation models. Here's why: %v\n", err)
  return
 }
 if len(result.ModelSummaries) == 0 {
  fmt.Println("There are no foundation models.")
 }
 for _, modelSummary := range result.ModelSummaries {
  fmt.Println(*modelSummary.ModelId)
 }
}
```

- For API details, see ListFoundationModels in *AWS SDK for Go API Reference*.

Actions

`ListFoundationModels`

The following code example shows how to use `ListFoundationModels`.

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.

List the available Bedrock foundation models.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/bedrock"
 "github.com/aws/aws-sdk-go-v2/service/bedrock/types"
)

// FoundationModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock service client that is used to perform foundation model
 actions.
type FoundationModelWrapper struct {
 BedrockClient *bedrock.Client
}



// ListPolicies lists Bedrock foundation models that you can use.
func (wrapper FoundationModelWrapper) ListFoundationModels(ctx context.Context)
 ([]types.FoundationModelSummary, error) {

 var models []types.FoundationModelSummary

 result, err := wrapper.BedrockClient.ListFoundationModels(ctx,
 &bedrock.ListFoundationModelsInput{})

 if err != nil {
  log.Printf("Couldn't list foundation models. Here's why: %v\n", err)
 } else {
  models = result.ModelSummaries
 }
 return models, err
}
```

- For API details, see ListFoundationModels in *AWS SDK for Go API Reference*.

Amazon Bedrock Runtime examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon Bedrock Runtime.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Amazon Nova
- Amazon Titan Image Generator
- Anthropic Claude

Get started

Hello Amazon Bedrock Runtime

The following code example shows how to get started using Amazon Bedrock Runtime.

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"
 "flag"
 "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/bedrockruntime"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime/types"
)

// main uses the AWS SDK for Go (v2) to create an Amazon Bedrock Runtime client
// and invokes Anthropic Claude using the Converse API.
// This example uses the default settings specified in your shared credentials
// and config files.
func main() {

 region := flag.String("region", "us-east-1", "The AWS region")
 flag.Parse()

 fmt.Printf("Using AWS region: %s\n", *region)

 ctx := context.Background()
 sdkConfig, err := config.LoadDefaultConfig(ctx, config.WithRegion(*region))
 if err != nil {
  fmt.Println("Couldn't load default configuration. Have you set up your AWS
 account?")
  fmt.Println(err)
  return
 }

 client := bedrockruntime.NewFromConfig(sdkConfig)

 // Set the model ID, e.g., Claude Haiku.
 // The "global." prefix enables cross-region inference, allowing the request
 // to be routed to the nearest available region for the specified model.
 modelId := "global.anthropic.claude-haiku-4-5-20251001-v1:0"

 // Start a conversation with the user message.
 prompt := "Hello. In a short paragraph, explain what you can do."

 message := types.Message{
  Content: []types.ContentBlock{
   &types.ContentBlockMemberText{Value: prompt},
  },
  Role: types.ConversationRoleUser,
 }

 fmt.Printf("Model: %s\n", modelId)
 fmt.Printf("Prompt: %s\n\n", prompt)
```

```
 result, err := client.Converse(ctx, &bedrockruntime.ConverseInput{
  ModelId:  aws.String(modelId),
  Messages: []types.Message{message},
 })

 if err != nil {
  log.Fatalf("ERROR: Can't invoke '%s'. Reason: %v\n", modelId, err)
 }

 // Extract and print the response text.
 responseMessage, ok := result.Output.(*types.ConverseOutputMemberMessage)
 if !ok {
  log.Fatal("ERROR: Unexpected output type from Converse API")
 }
 responseContentBlock := responseMessage.Value.Content[0]
 text, ok := responseContentBlock.(*types.ContentBlockMemberText)
 if !ok {
  log.Fatal("ERROR: Unexpected content block type from Converse API")
 }
 fmt.Printf("Response: %s\n", text.Value)
}
```

- For API details, see Converse in *AWS SDK for Go API Reference*.

Amazon Nova

Converse

The following code example shows how to send a text message to Amazon Nova, using Bedrock's Converse API.

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.

Send a text message to Amazon Nova, using Bedrock's Converse API.

```
func (wrapper ConverseWrapper) ConverseNova(ctx context.Context, prompt string)
 (string, error) {
 var content = types.ContentBlockMemberText{
  Value: prompt,
 }
 var message = types.Message{
  Content: []types.ContentBlock{&content},
  Role:    "user",
 }
 modelId := "amazon.nova-lite-v1:0"
 var converseInput = bedrockruntime.ConverseInput{
  ModelId:  aws.String(modelId),
  Messages: []types.Message{message},
 }
 response, err := wrapper.BedrockRuntimeClient.Converse(ctx, &converseInput)
 if err != nil {
  ProcessError(err, modelId)
 }

 responseText, _ := response.Output.(*types.ConverseOutputMemberMessage)
 responseContentBlock := responseText.Value.Content[0]
 text, _ := responseContentBlock.(*types.ContentBlockMemberText)
 return text.Value, nil
}
```

- For API details, see Converse in *AWS SDK for Go API Reference*.

Amazon Titan Image Generator

InvokeModel

The following code example shows how to invoke Amazon Titan Image on Amazon Bedrock to generate an image.

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.

Create an image with the Amazon Titan Image Generator.

```
import (
 "context"
 "encoding/json"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
)

// InvokeModelWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke foundation models.
type InvokeModelWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



type TitanImageRequest struct {
 TaskType              string                `json:"taskType"`
 TextToImageParams     TextToImageParams     `json:"textToImageParams"`
 ImageGenerationConfig ImageGenerationConfig `json:"imageGenerationConfig"`
}
type TextToImageParams struct {
 Text string `json:"text"`
}
type ImageGenerationConfig struct {
 NumberOfImages int     `json:"numberOfImages"`
 Quality        string  `json:"quality"`
 CfgScale       float64 `json:"cfgScale"`
 Height         int     `json:"height"`
 Width          int     `json:"width"`
```

```
 Seed           int64   `json:"seed"`
}

type TitanImageResponse struct {
 Images []string `json:"images"`
}

// Invokes the Titan Image model to create an image using the input provided
// in the request body.
func (wrapper InvokeModelWrapper) InvokeTitanImage(ctx context.Context, prompt
 string, seed int64) (string, error) {
 modelId := "amazon.titan-image-generator-v2:0"

 body, err := json.Marshal(TitanImageRequest{
  TaskType: "TEXT_IMAGE",
  TextToImageParams: TextToImageParams{
   Text: prompt,
  },
  ImageGenerationConfig: ImageGenerationConfig{
   NumberOfImages: 1,
   Quality:        "standard",
   CfgScale:       8.0,
   Height:         512,
   Width:          512,
   Seed:           seed,
  },
 })

 if err != nil {
  log.Fatal("failed to marshal", err)
 }

 output, err := wrapper.BedrockRuntimeClient.InvokeModel(ctx,
 &bedrockruntime.InvokeModelInput{
  ModelId:     aws.String(modelId),
  ContentType: aws.String("application/json"),
  Body:        body,
 })

 if err != nil {
  ProcessError(err, modelId)
 }

 var response TitanImageResponse
```

```
 if err := json.Unmarshal(output.Body, &response); err != nil {
  log.Fatal("failed to unmarshal", err)
 }

 base64ImageData := response.Images[0]

 return base64ImageData, nil

}
```

- For API details, see InvokeModel in *AWS SDK for Go API Reference*.

Anthropic Claude

Converse

The following code example shows how to send a text message to Anthropic Claude, using Bedrock's Converse API.

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.

Send a text message to Anthropic Claude, using Bedrock's Converse API.

```
import (
 "context"
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime"
 "github.com/aws/aws-sdk-go-v2/service/bedrockruntime/types"
)

// ConverseWrapper encapsulates Amazon Bedrock actions used in the examples.
// It contains a Bedrock Runtime client that is used to invoke Bedrock.
```

```
type ConverseWrapper struct {
 BedrockRuntimeClient *bedrockruntime.Client
}



func (wrapper ConverseWrapper) ConverseClaude(ctx context.Context, prompt string)
 (string, error) {
 var content = types.ContentBlockMemberText{
  Value: prompt,
 }
 var message = types.Message{
  Content: []types.ContentBlock{&content},
  Role:    "user",
 }
 // Claude Haiku 5.5 is available through an inference profile, so this uses
 // the "us." cross-Region profile ID.
 modelId := "us.anthropic.claude-haiku-5-5"
 var converseInput = bedrockruntime.ConverseInput{
  ModelId:  aws.String(modelId),
  Messages: []types.Message{message},
 }
 response, err := wrapper.BedrockRuntimeClient.Converse(ctx, &converseInput)
 if err != nil {
  ProcessError(err, modelId)
 }

 responseText, _ := response.Output.(*types.ConverseOutputMemberMessage)
 responseContentBlock := responseText.Value.Content[0]
 text, _ := responseContentBlock.(*types.ContentBlockMemberText)
 return text.Value, nil

}
```

- For API details, see Converse in *AWS SDK for Go API Reference*.

CloudFormation examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with CloudFormation.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`DescribeStacks`

The following code example shows how to use `DescribeStacks`.

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"
 "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
}
```

- For API details, see DescribeStacks in *AWS SDK for Go API Reference*.

CloudWatch Logs examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with CloudWatch Logs.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`StartLiveTail`

The following code example shows how to use `StartLiveTail`.

SDK for Go V2

Include the required files.

```
import (
 "context"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs"
 "github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs/types"
)
```

Handle the events from the Live Tail session.

```
func handleEventStreamAsync(stream *cloudwatchlogs.StartLiveTailEventStream) {
 eventsChan := stream.Events()
 for {
  event := <-eventsChan
  switch e := event.(type) {
  case *types.StartLiveTailResponseStreamMemberSessionStart:
   log.Println("Received SessionStart event")
  case *types.StartLiveTailResponseStreamMemberSessionUpdate:
   for _, logEvent := range e.Value.SessionResults {
    log.Println(*logEvent.Message)
   }
  default:
   // Handle on-stream exceptions
   if err := stream.Err(); err != nil {
    log.Fatalf("Error occured during streaming: %v", err)
   } else if event == nil {
    log.Println("Stream is Closed")
    return
   } else {
    log.Fatalf("Unknown event type: %T", e)
   }
  }
 }
}
```

Start the Live Tail session.

```
cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
 panic("configuration error, " + err.Error())
}
client := cloudwatchlogs.NewFromConfig(cfg)

request := &cloudwatchlogs.StartLiveTailInput{
 LogGroupIdentifiers:   logGroupIdentifiers,
 LogStreamNames:        logStreamNames,
 LogEventFilterPattern: logEventFilterPattern,
}

response, err := client.StartLiveTail(context.TODO(), request)
// Handle pre-stream Exceptions
if err != nil {
 log.Fatalf("Failed to start streaming: %v", err)
}

// Start a Goroutine to handle events over stream
stream := response.GetStream()
go handleEventStreamAsync(stream)
```

Stop the Live Tail session after a period of time has elapsed.

```
// Close the stream (which ends the session) after a timeout
time.Sleep(10 * time.Second)
stream.Close()
log.Println("Event stream closed")
```

- For API details, see StartLiveTail in *AWS SDK for Go API Reference*.

Amazon Cognito Identity Provider examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon Cognito Identity Provider.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Actions
- Scenarios

Get started

Hello Amazon Cognito

The following code example shows how to get started using Amazon Cognito.

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*.

Actions

`AdminCreateUser`

The following code example shows how to use `AdminCreateUser`.

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*.

`AdminSetUserPassword`

The following code example shows how to use `AdminSetUserPassword`.

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*.

`ConfirmForgotPassword`

The following code example shows how to use `ConfirmForgotPassword`.

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*.

`DeleteUser`

The following code example shows how to use `DeleteUser`.

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*.

`ForgotPassword`

The following code example shows how to use `ForgotPassword`.

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*.

`InitiateAuth`

The following code example shows how to use `InitiateAuth`.

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*.

`ListUserPools`

The following code example shows how to use `ListUserPools`.

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*.

`SignUp`

The following code example shows how to use `SignUp`.

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*.

`UpdateUserPool`

The following code example shows how to use `UpdateUserPool`.

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*.

Scenarios

Automatically confirm known users with a Lambda function

The following code example shows 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.

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

Automatically migrate known users with a Lambda function

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.

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

Write custom activity data with a Lambda function after Amazon Cognito user authentication

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.

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

Amazon DocumentDB examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon DocumentDB.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Serverless examples

Serverless examples

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Amazon DocumentDB event with Lambda using Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
```

```
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

DynamoDB examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with DynamoDB.

Basics are code examples that show you how to perform the essential operations within a service.

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.

AWS community contributions are examples that were created and are maintained by multiple teams across AWS. To provide feedback, use the mechanism provided in the linked repositories.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Basics
- Actions
- Scenarios
- Serverless examples
- AWS community contributions

Basics

Learn the basics

The following code example shows how to:

- Create a table that can hold movie data.
- Put, get, and update a single movie in the table.
- Write movie data to the table from a sample JSON file.
- Query for movies that were released in a given year.
- Scan for movies that were released in a range of years.
- Delete a movie from the table, then delete the table.

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 to create the table and perform actions on it.

```
import (
 "context"
```

```
 "fmt"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunMovieScenario is an interactive example that shows you how to use the AWS SDK
 for Go
// to create and use an Amazon DynamoDB table that stores data about movies.
//
//  1. Create a table that can hold movie data.
//  2. Put, get, and update a single movie in the table.
//  3. Write movie data to the table from a sample JSON file.
//  4. Query for movies that were released in a given year.
//  5. Scan for movies that were released in a range of years.
//  6. Delete a movie from the table.
//  7. Delete the table.
//
// This example creates a DynamoDB service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// The specified movie sampler is used to get sample data from a URL that is loaded
// into the named table.
func RunMovieScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner,
 tableName string,
 movieSampler actions.IMovieSampler) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB getting started demo.")
```

```
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{TableName: tableName,
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig)}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}

var customMovie actions.Movie
customMovie.Title = questioner.Ask("Enter a movie title to add to the table:",
 demotools.NotEmpty{})
customMovie.Year = questioner.AskInt("What year was it released?",
 demotools.NotEmpty{}, demotools.InIntRange{Lower: 1900, Upper: 2030})
customMovie.Info = map[string]interface{}{}
customMovie.Info["rating"] = questioner.AskFloat64(
 "Enter a rating between 1 and 10:",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
customMovie.Info["plot"] = questioner.Ask("What's the plot? ",
 demotools.NotEmpty{})
err = tableBasics.AddMovie(ctx, customMovie)
if err == nil {
 log.Printf("Added %v to the movie table.\n", customMovie.Title)
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's update your movie. You previously rated it %v.\n",
customMovie.Info["rating"])
customMovie.Info["rating"] = questioner.AskFloat64(
 "What new rating would you give it?",
 demotools.NotEmpty{}, demotools.InFloatRange{Lower: 1, Upper: 10})
log.Printf("You summarized the plot as '%v'.\n", customMovie.Info["plot"])
```

```
customMovie.Info["plot"] = questioner.Ask("What would you say now?",
 demotools.NotEmpty{})
attributes, err := tableBasics.UpdateMovie(ctx, customMovie)
if err == nil {
 log.Printf("Updated %v with new values.\n", customMovie.Title)
 for _, attVal := range attributes {
  for valKey, val := range attVal {
   log.Printf("\t%v: %v\n", valKey, val)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Getting movie data from %v and adding 250 movies to the table...\n",
 movieSampler.GetURL())
movies := movieSampler.GetSampleMovies()
written, err := tableBasics.AddMovieBatch(ctx, movies, 250)
if err != nil {
 panic(err)
} else {
 log.Printf("Added %v movies to the table.\n", written)
}

show := 10
if show > written {
 show = written
}
log.Printf("The first %v movies in the table are:", show)
for index, movie := range movies[:show] {
 log.Printf("\t%v. %v\n", index+1, movie.Title)
}
movieIndex := questioner.AskInt(
 "Enter the number of a movie to get info about it: ",
 demotools.InIntRange{Lower: 1, Upper: show},
)
movie, err := tableBasics.GetMovie(ctx, movies[movieIndex-1].Title,
movies[movieIndex-1].Year)
if err == nil {
 log.Println(movie)
}
log.Println(strings.Repeat("-", 88))

log.Println("Let's get a list of movies released in a given year.")
releaseYear := questioner.AskInt("Enter a year between 1972 and 2018: ",
```

```
 demotools.InIntRange{Lower: 1972, Upper: 2018},
)
releases, err := tableBasics.Query(ctx, releaseYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released in %v!\n", releaseYear)
 } else {
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

log.Println("Now let's scan for movies released in a range of years.")
startYear := questioner.AskInt("Enter a year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
endYear := questioner.AskInt("Enter another year: ",
 demotools.InIntRange{Lower: 1972, Upper: 2018})
releases, err = tableBasics.Scan(ctx, startYear, endYear)
if err == nil {
 if len(releases) == 0 {
  log.Printf("I couldn't find any movies released between %v and %v!\n", startYear,
endYear)
 } else {
  log.Printf("Found %v movies. In this list, the plot is <nil> because "+
   "we used a projection expression when scanning for items to return only "+
   "the title, year, and rating.\n", len(releases))
  for _, movie = range releases {
   log.Println(movie)
  }
 }
}
log.Println(strings.Repeat("-", 88))

var tables []string
if questioner.AskBool("Do you want to list all of your tables? (y/n) ", "y") {
 tables, err = tableBasics.ListTables(ctx)
 if err == nil {
  log.Printf("Found %v tables:", len(tables))
  for _, table := range tables {
   log.Printf("\t%v", table)
  }
 }
```

```
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Let's remove your movie '%v'.\n", customMovie.Title)
 if questioner.AskBool("Do you want to delete it from the table? (y/n) ", "y") {
  err = tableBasics.DeleteMovie(ctx, customMovie)
 }
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 if questioner.AskBool("Delete the table, too? (y/n)", "y") {
  err = tableBasics.DeleteTable(ctx)
 } else {
  log.Println("Don't forget to delete the table when you're done or you might " +
   "incur charges on your account.")
 }
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Create a struct and methods that call DynamoDB actions.

```
import (
 "context"
 "errors"
 "log"
 "time"
```

```
 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
```

```
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}



// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
```

```
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
```

```
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx, &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
```

```
   if err != nil {
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}



// GetMovie gets movie data from the DynamoDB table by using the primary composite
 key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
```

```
 return movie, err
}



// Query gets all movies in the DynamoDB table that were released in the specified
 year.
// The function uses the `expression` package to build the key condition expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
```

```
 return movies, err
}



// Scan gets all movies in the DynamoDB table that were released in a range of years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"), expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why: %v
\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
```

```
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - BatchWriteItem
  - CreateTable
  - DeleteItem
  - DeleteTable

- DescribeTable
- GetItem
- PutItem
- Query
- Scan
- UpdateItem

Actions

`BatchExecuteStatement`

The following code example shows how to use `BatchExecuteStatement`.

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.

Define a function receiver struct for the example.

```
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"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
```

```
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Use batches of INSERT statements to add items.

```
// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies to
 the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie) error
 {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Use batches of SELECT statements to get items.

```
// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}
```

Use batches of UPDATE statements to update items.

```
// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the rating
 of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Use batches of DELETE statements to delete items.

```
// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
```

```
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
```

```
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see BatchExecuteStatement in *AWS SDK for Go API Reference*.

`BatchWriteItem`

The following code example shows how to use `BatchWriteItem`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch adds a slice of movies to the DynamoDB table. The function sends
// batches of 25 movies to DynamoDB until all movies are added or it reaches the
// specified maximum.
func (basics TableBasics) AddMovieBatch(ctx context.Context, movies []Movie,
 maxMovies int) (int, error) {
 var err error
 var item map[string]types.AttributeValue
 written := 0
 batchSize := 25 // DynamoDB allows a maximum batch size of 25 items.
 start := 0
 end := start + batchSize
 for start < maxMovies && start < len(movies) {
  var writeReqs []types.WriteRequest
  if end > len(movies) {
   end = len(movies)
  }
  for _, movie := range movies[start:end] {
   item, err = attributevalue.MarshalMap(movie)
   if err != nil {
```

```
    log.Printf("Couldn't marshal movie %v for batch writing. Here's why: %v\n",
 movie.Title, err)
   } else {
    writeReqs = append(
     writeReqs,
     types.WriteRequest{PutRequest: &types.PutRequest{Item: item}},
    )
   }
  }
  _, err = basics.DynamoDbClient.BatchWriteItem(ctx, &dynamodb.BatchWriteItemInput{
   RequestItems: map[string][]types.WriteRequest{basics.TableName: writeReqs}})
  if err != nil {
   log.Printf("Couldn't add a batch of movies to %v. Here's why: %v\n",
 basics.TableName, err)
  } else {
   written += len(writeReqs)
  }
  start = end
  end += batchSize
 }

 return written, err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)
```

```
// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see BatchWriteItem in *AWS SDK for Go API Reference*.

`CreateTable`

The following code example shows how to use `CreateTable`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// CreateMovieTable creates a DynamoDB table with a composite primary key defined as
// a string sort key named `title`, and a numeric partition key named `year`.
// This function uses NewTableExistsWaiter to wait for the table to be created by
// DynamoDB before it returns.
func (basics TableBasics) CreateMovieTable(ctx context.Context)
 (*types.TableDescription, error) {
 var tableDesc *types.TableDescription
 table, err := basics.DynamoDbClient.CreateTable(ctx, &dynamodb.CreateTableInput{
  AttributeDefinitions: []types.AttributeDefinition{{
   AttributeName: aws.String("year"),
```

```
   AttributeType: types.ScalarAttributeTypeN,
  }, {
   AttributeName: aws.String("title"),
   AttributeType: types.ScalarAttributeTypeS,
  }},
  KeySchema: []types.KeySchemaElement{{
   AttributeName: aws.String("year"),
   KeyType:       types.KeyTypeHash,
  }, {
   AttributeName: aws.String("title"),
   KeyType:       types.KeyTypeRange,
  }},
  TableName:   aws.String(basics.TableName),
  BillingMode: types.BillingModePayPerRequest,
 })
 if err != nil {
  log.Printf("Couldn't create table %v. Here's why: %v\n", basics.TableName, err)
 } else {
  waiter := dynamodb.NewTableExistsWaiter(basics.DynamoDbClient)
  err = waiter.Wait(ctx, &dynamodb.DescribeTableInput{
   TableName: aws.String(basics.TableName)}, 5*time.Minute)
  if err != nil {
   log.Printf("Wait for table exists failed. Here's why: %v\n", err)
  }
  tableDesc = table.TableDescription
  log.Printf("Ccreating table test")
 }
 return tableDesc, err
}
```

- For API details, see CreateTable in *AWS SDK for Go API Reference*.

`DeleteItem`

The following code example shows how to use `DeleteItem`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteMovie removes a movie from the DynamoDB table.
func (basics TableBasics) DeleteMovie(ctx context.Context, movie Movie) error {
 _, err := basics.DynamoDbClient.DeleteItem(ctx, &dynamodb.DeleteItemInput{
  TableName: aws.String(basics.TableName), Key: movie.GetKey(),
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
```

```
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
```

```
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see DeleteItem in *AWS SDK for Go API Reference*.

`DeleteTable`

The following code example shows how to use `DeleteTable`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
```

```
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// DeleteTable deletes the DynamoDB table and all of its data.
func (basics TableBasics) DeleteTable(ctx context.Context) error {
 _, err := basics.DynamoDbClient.DeleteTable(ctx, &dynamodb.DeleteTableInput{
  TableName: aws.String(basics.TableName)})
 if err != nil {
  log.Printf("Couldn't delete table %v. Here's why: %v\n", basics.TableName, err)
 }
 return err
}
```

- For API details, see DeleteTable in *AWS SDK for Go API Reference*.

`DescribeTable`

The following code example shows how to use `DescribeTable`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// TableExists determines whether a DynamoDB table exists.
func (basics TableBasics) TableExists(ctx context.Context) (bool, error) {
 exists := true
 _, err := basics.DynamoDbClient.DescribeTable(
  ctx, &dynamodb.DescribeTableInput{TableName: aws.String(basics.TableName)},
 )
 if err != nil {
  var notFoundEx *types.ResourceNotFoundException
  if errors.As(err, &notFoundEx) {
   log.Printf("Table %v does not exist.\n", basics.TableName)
   err = nil
  } else {
   log.Printf("Couldn't determine existence of table %v. Here's why: %v\n",
 basics.TableName, err)
  }
  exists = false
 }
 return exists, err
}
```

- For API details, see DescribeTable in *AWS SDK for Go API Reference*.

`ExecuteStatement`

The following code example shows how to use `ExecuteStatement`.

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.

Define a function receiver struct for the example.

```
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"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

Use an INSERT statement to add an item.

```
// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
```

```
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}
```

Use a SELECT statement to get an item.

```
// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB table
 by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
```

```
 }
 return movie, err
}
```

Use a SELECT statement to get a list of items and project the results.

```
// GetAllMovies runs a PartiQL SELECT statement to get all movies from the DynamoDB
 table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
   nextToken = response.NextToken
   moreData = nextToken != nil
  }
```

```
 }
 return output, err
}
```

Use an UPDATE statement to update an item.

```
// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie that
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}
```

Use a DELETE statement to delete an item.

```
// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the DynamoDB
 table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
 if err != nil {
  panic(err)
 }
```

```
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}
```

```
// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see ExecuteStatement in *AWS SDK for Go API Reference*.

`GetItem`

The following code example shows how to use `GetItem`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// GetMovie gets movie data from the DynamoDB table by using the primary composite
 key
// made of title and year.
func (basics TableBasics) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 movie := Movie{Title: title, Year: year}
 response, err := basics.DynamoDbClient.GetItem(ctx, &dynamodb.GetItemInput{
  Key: movie.GetKey(), TableName: aws.String(basics.TableName),
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Item, &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see GetItem in *AWS SDK for Go API Reference*.

`ListTables`

The following code example shows how to use `ListTables`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}
```

```
// ListTables lists the DynamoDB table names for the current account.
func (basics TableBasics) ListTables(ctx context.Context) ([]string, error) {
 var tableNames []string
 var output *dynamodb.ListTablesOutput
 var err error
 tablePaginator := dynamodb.NewListTablesPaginator(basics.DynamoDbClient,
 &dynamodb.ListTablesInput{})
 for tablePaginator.HasMorePages() {
  output, err = tablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't list tables. Here's why: %v\n", err)
   break
  } else {
   tableNames = append(tableNames, output.TableNames...)
  }
 }
 return tableNames, err
}
```

- For API details, see ListTables in *AWS SDK for Go API Reference*.

`PutItem`

The following code example shows how to use `PutItem`.

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"
 "time"
```

```
 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie adds a movie the DynamoDB table.
func (basics TableBasics) AddMovie(ctx context.Context, movie Movie) error {
 item, err := attributevalue.MarshalMap(movie)
 if err != nil {
  panic(err)
 }
 _, err = basics.DynamoDbClient.PutItem(ctx, &dynamodb.PutItemInput{
  TableName: aws.String(basics.TableName), Item: item,
 })
 if err != nil {
  log.Printf("Couldn't add item to table. Here's why: %v\n", err)
 }
 return err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
```

```
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see PutItem in *AWS SDK for Go API Reference*.

`Query`

The following code example shows how to use `Query`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Query gets all movies in the DynamoDB table that were released in the specified
 year.
// The function uses the `expression` package to build the key condition expression
// that is used in the query.
func (basics TableBasics) Query(ctx context.Context, releaseYear int) ([]Movie,
 error) {
 var err error
```

```
 var response *dynamodb.QueryOutput
 var movies []Movie
 keyEx := expression.Key("year").Equal(expression.Value(releaseYear))
 expr, err := expression.NewBuilder().WithKeyCondition(keyEx).Build()
 if err != nil {
  log.Printf("Couldn't build expression for query. Here's why: %v\n", err)
 } else {
  queryPaginator := dynamodb.NewQueryPaginator(basics.DynamoDbClient,
 &dynamodb.QueryInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   KeyConditionExpression:    expr.KeyCondition(),
  })
  for queryPaginator.HasMorePages() {
   response, err = queryPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't query for movies released in %v. Here's why: %v\n",
 releaseYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
```

```
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see Query in *AWS SDK for Go API Reference*.

Scan

The following code example shows how to use Scan.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// Scan gets all movies in the DynamoDB table that were released in a range of years
// and projects them to return a reduced set of fields.
// The function uses the `expression` package to build the filter and projection
// expressions.
```

```
func (basics TableBasics) Scan(ctx context.Context, startYear int, endYear int)
 ([]Movie, error) {
 var movies []Movie
 var err error
 var response *dynamodb.ScanOutput
 filtEx := expression.Name("year").Between(expression.Value(startYear),
 expression.Value(endYear))
 projEx := expression.NamesList(
  expression.Name("year"), expression.Name("title"), expression.Name("info.rating"))
 expr, err :=
 expression.NewBuilder().WithFilter(filtEx).WithProjection(projEx).Build()
 if err != nil {
  log.Printf("Couldn't build expressions for scan. Here's why: %v\n", err)
 } else {
  scanPaginator := dynamodb.NewScanPaginator(basics.DynamoDbClient,
 &dynamodb.ScanInput{
   TableName:                 aws.String(basics.TableName),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   FilterExpression:          expr.Filter(),
   ProjectionExpression:      expr.Projection(),
  })
  for scanPaginator.HasMorePages() {
   response, err = scanPaginator.NextPage(ctx)
   if err != nil {
    log.Printf("Couldn't scan for movies released between %v and %v. Here's why: %v
\n",
     startYear, endYear, err)
    break
   } else {
    var moviePage []Movie
    err = attributevalue.UnmarshalListOfMaps(response.Items, &moviePage)
    if err != nil {
     log.Printf("Couldn't unmarshal query response. Here's why: %v\n", err)
     break
    } else {
     movies = append(movies, moviePage...)
    }
   }
  }
 }
 return movies, err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see Scan in *AWS SDK for Go API Reference*.

`UpdateItem`

The following code example shows how to use `UpdateItem`.

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"
 "time"

 "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/feature/dynamodb/expression"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// TableBasics encapsulates the Amazon DynamoDB service actions used in the
 examples.
// It contains a DynamoDB service client that is used to act on the specified table.
type TableBasics struct {
 DynamoDbClient *dynamodb.Client
```

```
 TableName      string
}



// UpdateMovie updates the rating and plot of a movie that already exists in the
// DynamoDB table. This function uses the `expression` package to build the update
// expression.
func (basics TableBasics) UpdateMovie(ctx context.Context, movie Movie)
 (map[string]map[string]interface{}, error) {
 var err error
 var response *dynamodb.UpdateItemOutput
 var attributeMap map[string]map[string]interface{}
 update := expression.Set(expression.Name("info.rating"),
 expression.Value(movie.Info["rating"]))
 update.Set(expression.Name("info.plot"), expression.Value(movie.Info["plot"]))
 expr, err := expression.NewBuilder().WithUpdate(update).Build()
 if err != nil {
  log.Printf("Couldn't build expression for update. Here's why: %v\n", err)
 } else {
  response, err = basics.DynamoDbClient.UpdateItem(ctx, &dynamodb.UpdateItemInput{
   TableName:                 aws.String(basics.TableName),
   Key:                       movie.GetKey(),
   ExpressionAttributeNames:  expr.Names(),
   ExpressionAttributeValues: expr.Values(),
   UpdateExpression:          expr.Update(),
   ReturnValues:              types.ReturnValueUpdatedNew,
  })
  if err != nil {
   log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
  } else {
   err = attributevalue.UnmarshalMap(response.Attributes, &attributeMap)
   if err != nil {
    log.Printf("Couldn't unmarshall update response. Here's why: %v\n", err)
   }
  }
 }
 return attributeMap, err
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
```

```
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

- For API details, see UpdateItem in *AWS SDK for Go API Reference*.

Scenarios

Query a table by using batches of PartiQL statements

The following code example shows how to:

- Get a batch of items by running multiple SELECT statements.
- Add a batch of items by running multiple INSERT statements.
- Update a batch of items by running multiple UPDATE statements.
- Delete a batch of items by running multiple DELETE statements.

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 a scenario that creates a table and runs batches of PartiQL queries.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)
```

```
// RunPartiQLBatchScenario shows you how to use the AWS SDK for Go
// to run batches of PartiQL statements to query a table that stores data about
 movies.
//
//   - Use batches of PartiQL statements to add, get, update, and delete data for
//     individual movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLBatchScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon DynamoDB PartiQL batch demo.")
 log.Println(strings.Repeat("-", 88))

 tableBasics := actions.TableBasics{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }
 runner := actions.PartiQLRunner{
  DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
  TableName:      tableName,
 }

 exists, err := tableBasics.TableExists(ctx)
 if err != nil {
  panic(err)
 }
 if !exists {
  log.Printf("Creating table %v...\n", tableName)
  _, err = tableBasics.CreateMovieTable(ctx)
  if err != nil {
   panic(err)
  } else {
```

```
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovies := []actions.Movie{{
 Title: "House PartiQL",
 Year:  currentYear - 5,
 Info: map[string]interface{}{
  "plot":   "Wacky high jinks result from querying a mysterious database.",
  "rating": 8.5}}, {
 Title: "House PartiQL 2",
 Year:  currentYear - 3,
 Info: map[string]interface{}{
  "plot":   "Moderate high jinks result from querying another mysterious
database.",
  "rating": 6.5}}, {
 Title: "House PartiQL 3",
 Year:  currentYear - 1,
 Info: map[string]interface{}{
  "plot":   "Tepid high jinks result from querying yet another mysterious
database.",
  "rating": 2.5},
},
}

log.Printf("Inserting a batch of movies into table '%v'.\n", tableName)
err = runner.AddMovieBatch(ctx, customMovies)
if err == nil {
 log.Printf("Added %v movies to the table.\n", len(customMovies))
}
log.Println(strings.Repeat("-", 88))

log.Println("Getting data for a batch of movies.")
movies, err := runner.GetMovieBatch(ctx, customMovies)
if err == nil {
 for _, movie := range movies {
  log.Println(movie)
 }
}
log.Println(strings.Repeat("-", 88))
```

```
 newRatings := []float64{7.7, 4.4, 1.1}
 log.Println("Updating a batch of movies with new ratings.")
 err = runner.UpdateMovieBatch(ctx, customMovies, newRatings)
 if err == nil {
  log.Printf("Updated %v movies with new ratings.\n", len(customMovies))
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Getting projected data from the table to verify our update.")
 log.Println("Using a page size of 2 to demonstrate paging.")
 projections, err := runner.GetAllMovies(ctx, 2)
 if err == nil {
  log.Println("All movies:")
  for _, projection := range projections {
   log.Println(projection)
  }
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Deleting a batch of movies.")
 err = runner.DeleteMovieBatch(ctx, customMovies)
 if err == nil {
  log.Printf("Deleted %v movies.\n", len(customMovies))
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
```

```
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                 `dynamodbav:"title"`
 Year  int                    `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}

// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Create a struct and methods that run PartiQL statements.

```
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"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovieBatch runs a batch of PartiQL INSERT statements to add multiple movies to
 the
// DynamoDB table.
func (runner PartiQLRunner) AddMovieBatch(ctx context.Context, movies []Movie) error
 {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(fmt.Sprintf(
    "INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
 runner.TableName)),
```

```
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't insert a batch of items with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovieBatch runs a batch of PartiQL SELECT statements to get multiple movies
 from
// the DynamoDB table by title and year.
func (runner PartiQLRunner) GetMovieBatch(ctx context.Context, movies []Movie)
 ([]Movie, error) {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 output, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 var outMovies []Movie
 if err != nil {
  log.Printf("Couldn't get a batch of items with PartiQL. Here's why: %v\n", err)
 } else {
  for _, response := range output.Responses {
   var movie Movie
```

```
   err = attributevalue.UnmarshalMap(response.Item, &movie)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    outMovies = append(outMovies, movie)
   }
  }
 }
 return outMovies, err
}



// GetAllMovies runs a PartiQL SELECT statement to get all movies from the DynamoDB
 table.
// pageSize is not typically required and is used to show how to paginate the
 results.
// The results are projected to return only the title and rating of each movie.
func (runner PartiQLRunner) GetAllMovies(ctx context.Context, pageSize int32)
 ([]map[string]interface{}, error) {
 var output []map[string]interface{}
 var response *dynamodb.ExecuteStatementOutput
 var err error
 var nextToken *string
 for moreData := true; moreData; {
  response, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
   Statement: aws.String(
    fmt.Sprintf("SELECT title, info.rating FROM \"%v\"", runner.TableName)),
   Limit:     aws.Int32(pageSize),
   NextToken: nextToken,
  })
  if err != nil {
   log.Printf("Couldn't get movies. Here's why: %v\n", err)
   moreData = false
  } else {
   var pageOutput []map[string]interface{}
   err = attributevalue.UnmarshalListOfMaps(response.Items, &pageOutput)
   if err != nil {
    log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
   } else {
    log.Printf("Got a page of length %v.\n", len(response.Items))
    output = append(output, pageOutput...)
   }
```

```
    nextToken = response.NextToken
    moreData = nextToken != nil
   }
  }
  return output, err
}



// UpdateMovieBatch runs a batch of PartiQL UPDATE statements to update the rating
 of
// multiple movies that already exist in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovieBatch(ctx context.Context, movies []Movie,
 ratings []float64) error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{ratings[index],
 movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
 runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't update the batch of movies. Here's why: %v\n", err)
 }
 return err
}



// DeleteMovieBatch runs a batch of PartiQL DELETE statements to remove multiple
 movies
// from the DynamoDB table.
```

```
func (runner PartiQLRunner) DeleteMovieBatch(ctx context.Context, movies []Movie)
 error {
 statementRequests := make([]types.BatchStatementRequest, len(movies))
 for index, movie := range movies {
  params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
  if err != nil {
   panic(err)
  }
  statementRequests[index] = types.BatchStatementRequest{
   Statement: aws.String(
    fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?", runner.TableName)),
   Parameters: params,
  }
 }

 _, err := runner.DynamoDbClient.BatchExecuteStatement(ctx,
 &dynamodb.BatchExecuteStatementInput{
  Statements: statementRequests,
 })
 if err != nil {
  log.Printf("Couldn't delete the batch of movies. Here's why: %v\n", err)
 }
 return err
}
```

- For API details, see BatchExecuteStatement in *AWS SDK for Go API Reference*.

Query a table using PartiQL

The following code example shows how to:

- Get an item by running a SELECT statement.
- Add an item by running an INSERT statement.
- Update an item by running an UPDATE statement.
- Delete an item by running a DELETE statement.

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 a scenario that creates a table and runs PartiQL queries.

```
import (
 "context"
 "fmt"
 "log"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/dynamodb/actions"
)

// RunPartiQLSingleScenario shows you how to use the AWS SDK for Go
// to use PartiQL to query a table that stores data about movies.
//
// * Use PartiQL statements to add, get, update, and delete data for individual
 movies.
//
// This example creates an Amazon DynamoDB service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// This example creates and deletes a DynamoDB table to use during the scenario.
func RunPartiQLSingleScenario(ctx context.Context, sdkConfig aws.Config, tableName
 string) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Printf("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
```

```
log.Println("Welcome to the Amazon DynamoDB PartiQL single action demo.")
log.Println(strings.Repeat("-", 88))

tableBasics := actions.TableBasics{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}
runner := actions.PartiQLRunner{
 DynamoDbClient: dynamodb.NewFromConfig(sdkConfig),
 TableName:      tableName,
}

exists, err := tableBasics.TableExists(ctx)
if err != nil {
 panic(err)
}
if !exists {
 log.Printf("Creating table %v...\n", tableName)
 _, err = tableBasics.CreateMovieTable(ctx)
 if err != nil {
  panic(err)
 } else {
  log.Printf("Created table %v.\n", tableName)
 }
} else {
 log.Printf("Table %v already exists.\n", tableName)
}
log.Println(strings.Repeat("-", 88))

currentYear, _, _ := time.Now().Date()
customMovie := actions.Movie{
 Title: "24 Hour PartiQL People",
 Year:  currentYear,
 Info: map[string]interface{}{
  "plot":   "A group of data developers discover a new query language they can't
stop using.",
  "rating": 9.9,
 },
}

log.Printf("Inserting movie '%v' released in %v.", customMovie.Title,
customMovie.Year)
err = runner.AddMovie(ctx, customMovie)
if err == nil {
```

```
  log.Printf("Added %v to the movie table.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data for movie '%v' released in %v.", customMovie.Title,
 customMovie.Year)
 movie, err := runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 newRating := 6.6
 log.Printf("Updating movie '%v' with a rating of %v.", customMovie.Title,
 newRating)
 err = runner.UpdateMovie(ctx, customMovie, newRating)
 if err == nil {
  log.Printf("Updated %v with a new rating.\n", customMovie.Title)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Getting data again to verify the update.")
 movie, err = runner.GetMovie(ctx, customMovie.Title, customMovie.Year)
 if err == nil {
  log.Println(movie)
 }
 log.Println(strings.Repeat("-", 88))

 log.Printf("Deleting movie '%v'.\n", customMovie.Title)
 err = runner.DeleteMovie(ctx, customMovie)
 if err == nil {
  log.Printf("Deleted %v.\n", customMovie.Title)
 }

 err = tableBasics.DeleteTable(ctx)
 if err == nil {
  log.Printf("Deleted table %v.\n", tableBasics.TableName)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a Movie struct that is used in this example.

```
import (
 "archive/zip"
 "bytes"
 "encoding/json"
 "fmt"
 "io"
 "log"
 "net/http"

 "github.com/aws/aws-sdk-go-v2/feature/dynamodb/attributevalue"
 "github.com/aws/aws-sdk-go-v2/service/dynamodb/types"
)

// Movie encapsulates data about a movie. Title and Year are the composite primary
 key
// of the movie in Amazon DynamoDB. Title is the sort key, Year is the partition
 key,
// and Info is additional data.
type Movie struct {
 Title string                `dynamodbav:"title"`
 Year  int                   `dynamodbav:"year"`
 Info  map[string]interface{} `dynamodbav:"info"`
}

// GetKey returns the composite primary key of the movie in a format that can be
// sent to DynamoDB.
func (movie Movie) GetKey() map[string]types.AttributeValue {
 title, err := attributevalue.Marshal(movie.Title)
 if err != nil {
  panic(err)
 }
 year, err := attributevalue.Marshal(movie.Year)
 if err != nil {
  panic(err)
 }
 return map[string]types.AttributeValue{"title": title, "year": year}
}
```

```
// String returns the title, year, rating, and plot of a movie, formatted for the
 example.
func (movie Movie) String() string {
 return fmt.Sprintf("%v\n\tReleased: %v\n\tRating: %v\n\tPlot: %v\n",
  movie.Title, movie.Year, movie.Info["rating"], movie.Info["plot"])
}
```

Create a struct and methods that run PartiQL statements.

```
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"
)

// PartiQLRunner encapsulates the Amazon DynamoDB service actions used in the
// PartiQL examples. It contains a DynamoDB service client that is used to act on
 the
// specified table.
type PartiQLRunner struct {
 DynamoDbClient *dynamodb.Client
 TableName      string
}



// AddMovie runs a PartiQL INSERT statement to add a movie to the DynamoDB table.
func (runner PartiQLRunner) AddMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year,
 movie.Info})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
```

```
  Statement: aws.String(
   fmt.Sprintf("INSERT INTO \"%v\" VALUE {'title': ?, 'year': ?, 'info': ?}",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't insert an item with PartiQL. Here's why: %v\n", err)
 }
 return err
}



// GetMovie runs a PartiQL SELECT statement to get a movie from the DynamoDB table
 by
// title and year.
func (runner PartiQLRunner) GetMovie(ctx context.Context, title string, year int)
 (Movie, error) {
 var movie Movie
 params, err := attributevalue.MarshalList([]interface{}{title, year})
 if err != nil {
  panic(err)
 }
 response, err := runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("SELECT * FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't get info about %v. Here's why: %v\n", title, err)
 } else {
  err = attributevalue.UnmarshalMap(response.Items[0], &movie)
  if err != nil {
   log.Printf("Couldn't unmarshal response. Here's why: %v\n", err)
  }
 }
 return movie, err
}



// UpdateMovie runs a PartiQL UPDATE statement to update the rating of a movie that
```

```
// already exists in the DynamoDB table.
func (runner PartiQLRunner) UpdateMovie(ctx context.Context, movie Movie, rating
 float64) error {
 params, err := attributevalue.MarshalList([]interface{}{rating, movie.Title,
 movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("UPDATE \"%v\" SET info.rating=? WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't update movie %v. Here's why: %v\n", movie.Title, err)
 }
 return err
}



// DeleteMovie runs a PartiQL DELETE statement to remove a movie from the DynamoDB
 table.
func (runner PartiQLRunner) DeleteMovie(ctx context.Context, movie Movie) error {
 params, err := attributevalue.MarshalList([]interface{}{movie.Title, movie.Year})
 if err != nil {
  panic(err)
 }
 _, err = runner.DynamoDbClient.ExecuteStatement(ctx,
 &dynamodb.ExecuteStatementInput{
  Statement: aws.String(
   fmt.Sprintf("DELETE FROM \"%v\" WHERE title=? AND year=?",
    runner.TableName)),
  Parameters: params,
 })
 if err != nil {
  log.Printf("Couldn't delete %v from the table. Here's why: %v\n", movie.Title,
 err)
 }
 return err
}
```

- For API details, see ExecuteStatement in *AWS SDK for Go API Reference*.

Serverless examples

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a DynamoDB event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string, error)
 {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
```

```
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
}
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting DynamoDB batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)
```

```
type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*BatchResult,
 error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

AWS community contributions

Build and test a serverless application

The following code example shows how to build and test a serverless application using API Gateway with Lambda and DynamoDB

SDK for Go V2

Shows how to build and test a serverless application that consists of an API Gateway with Lambda and DynamoDB using the Go SDK.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- API Gateway
- DynamoDB
- Lambda

IAM examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with IAM.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions

Get started

Hello IAM

The following code example shows how to get started using IAM.

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"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/iam"
)

// main uses the AWS SDK for Go (v2) to create an AWS Identity and Access Management
 (IAM)
// client and list up to 10 policies 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
 }
 iamClient := iam.NewFromConfig(sdkConfig)
 const maxPols = 10
 fmt.Printf("Let's list up to %v policies for your account.\n", maxPols)
 result, err := iamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPols),
 })
 if err != nil {
  fmt.Printf("Couldn't list policies for your account. Here's why: %v\n", err)
```

```
  return
 }
 if len(result.Policies) == 0 {
  fmt.Println("You don't have any policies!")
 } else {
  for _, policy := range result.Policies {
   fmt.Printf("\t%v\n", *policy.PolicyName)
  }
 }
}
```

- For API details, see ListPolicies in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to create a user and assume a role.

> **Warning**
>
> To avoid security risks, don't use IAM users for authentication when developing purpose-built software or working with real data. Instead, use federation with an identity provider such as AWS IAM Identity Center.

- Create a user with no permissions.
- Create a role that grants permission to list Amazon S3 buckets for the account.
- Add a policy to let the user assume the role.
- Assume the role and list S3 buckets using temporary credentials, then clean up resources.

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"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/credentials"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/sts"
 "github.com/aws/smithy-go"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/iam/actions"
)

// AssumeRoleScenario shows you how to use the AWS Identity and Access Management
 (IAM)
// service to perform the following actions:
//
//  1. Create a user who has no permissions.
//  2. Create a role that grants permission to list Amazon Simple Storage Service
//     (Amazon S3) buckets for the account.
//  3. Add a policy to let the user assume the role.
//  4. Try and fail to list buckets without permissions.
//  5. Assume the role and list S3 buckets using temporary credentials.
//  6. Delete the policy, role, and user.
```

```
type AssumeRoleScenario struct {
 sdkConfig      aws.Config
 accountWrapper actions.AccountWrapper
 policyWrapper  actions.PolicyWrapper
 roleWrapper    actions.RoleWrapper
 userWrapper    actions.UserWrapper
 questioner     demotools.IQuestioner
 helper         IScenarioHelper
 isTestRun      bool
}

// NewAssumeRoleScenario constructs an AssumeRoleScenario instance from a
 configuration.
// It uses the specified config to get an IAM client and create wrappers for the
 actions
// used in the scenario.
func NewAssumeRoleScenario(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) AssumeRoleScenario {
 iamClient := iam.NewFromConfig(sdkConfig)
 return AssumeRoleScenario{
  sdkConfig:      sdkConfig,
  accountWrapper: actions.AccountWrapper{IamClient: iamClient},
  policyWrapper:  actions.PolicyWrapper{IamClient: iamClient},
  roleWrapper:    actions.RoleWrapper{IamClient: iamClient},
  userWrapper:    actions.UserWrapper{IamClient: iamClient},
  questioner:     questioner,
  helper:         helper,
 }
}

// addTestOptions appends the API options specified in the original configuration to
// another configuration. This is used to attach the middleware stubber to clients
// that are constructed during the scenario, which is needed for unit testing.
func (scenario AssumeRoleScenario) addTestOptions(scenarioConfig *aws.Config) {
 if scenario.isTestRun {
  scenarioConfig.APIOptions = append(scenarioConfig.APIOptions,
 scenario.sdkConfig.APIOptions...)
 }
}

// Run runs the interactive scenario.
func (scenario AssumeRoleScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
```

```
    log.Printf("Something went wrong with the demo.\n")
    log.Println(r)
   }
  }()

  log.Println(strings.Repeat("-", 88))
  log.Println("Welcome to the AWS Identity and Access Management (IAM) assume role
  demo.")
  log.Println(strings.Repeat("-", 88))

  user := scenario.CreateUser(ctx)
  accessKey := scenario.CreateAccessKey(ctx, user)
  role := scenario.CreateRoleAndPolicies(ctx, user)
  noPermsConfig := scenario.ListBucketsWithoutPermissions(ctx, accessKey)
  scenario.ListBucketsWithAssumedRole(ctx, noPermsConfig, role)
  scenario.Cleanup(ctx, user, role)

  log.Println(strings.Repeat("-", 88))
  log.Println("Thanks for watching!")
  log.Println(strings.Repeat("-", 88))
}

// CreateUser creates a new IAM user. This user has no permissions.
func (scenario AssumeRoleScenario) CreateUser(ctx context.Context) *types.User {
  log.Println("Let's create an example user with no permissions.")
  userName := scenario.questioner.Ask("Enter a name for the example user:",
  demotools.NotEmpty{})
  user, err := scenario.userWrapper.GetUser(ctx, userName)
  if err != nil {
   panic(err)
  }
  if user == nil {
   user, err = scenario.userWrapper.CreateUser(ctx, userName)
   if err != nil {
    panic(err)
   }
   log.Printf("Created user %v.\n", *user.UserName)
  } else {
   log.Printf("User %v already exists.\n", *user.UserName)
  }
  log.Println(strings.Repeat("-", 88))
  return user
}
```

```
// CreateAccessKey creates an access key for the user.
func (scenario AssumeRoleScenario) CreateAccessKey(ctx context.Context, user
 *types.User) *types.AccessKey {
 accessKey, err := scenario.userWrapper.CreateAccessKeyPair(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 log.Printf("Created access key %v for your user.", *accessKey.AccessKeyId)
 log.Println("Waiting a few seconds for your user to be ready...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return accessKey
}

// CreateRoleAndPolicies creates a policy that grants permission to list S3 buckets
 for
// the current account and attaches the policy to a newly created role. It also adds
 an
// inline policy to the specified user that grants the user permission to assume the
 role.
func (scenario AssumeRoleScenario) CreateRoleAndPolicies(ctx context.Context, user
 *types.User) *types.Role {
 log.Println("Let's create a role and policy that grant permission to list S3
 buckets.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 listBucketsRole, err := scenario.roleWrapper.CreateRole(ctx,
 scenario.helper.GetName(), *user.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created role %v.\n", *listBucketsRole.RoleName)
 listBucketsPolicy, err := scenario.policyWrapper.CreatePolicy(
  ctx, scenario.helper.GetName(), []string{"s3:ListAllMyBuckets"}, "arn:aws:s3:::*")
 if err != nil {
  panic(err)
 }
 log.Printf("Created policy %v.\n", *listBucketsPolicy.PolicyName)
 err = scenario.roleWrapper.AttachRolePolicy(ctx, *listBucketsPolicy.Arn,
 *listBucketsRole.RoleName)
 if err != nil {
  panic(err)
 }
 log.Printf("Attached policy %v to role %v.\n", *listBucketsPolicy.PolicyName,
  *listBucketsRole.RoleName)
```

```
 err = scenario.userWrapper.CreateUserPolicy(ctx, *user.UserName,
 scenario.helper.GetName(),
  []string{"sts:AssumeRole"}, *listBucketsRole.Arn)
 if err != nil {
  panic(err)
 }
 log.Printf("Created an inline policy for user %v that lets the user assume the
 role.\n",
  *user.UserName)
 log.Println("Let's give AWS a few seconds to propagate these new resources and
 connections...")
 scenario.helper.Pause(10)
 log.Println(strings.Repeat("-", 88))
 return listBucketsRole
}

// ListBucketsWithoutPermissions creates an Amazon S3 client from the user's access
 key
// credentials and tries to list buckets for the account. Because the user does not
 have
// permission to perform this action, the action fails.
func (scenario AssumeRoleScenario) ListBucketsWithoutPermissions(ctx
 context.Context, accessKey *types.AccessKey) *aws.Config {
 log.Println("Let's try to list buckets without permissions. This should return an
 AccessDenied error.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 noPermsConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *accessKey.AccessKeyId, *accessKey.SecretAccessKey, ""),
  ))
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&noPermsConfig)

 s3Client := s3.NewFromConfig(noPermsConfig)
 _, err = s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  // The SDK for Go does not model the AccessDenied error, so check ErrorCode
 directly.
  var ae smithy.APIError
```

```
  if errors.As(err, &ae) {
   switch ae.ErrorCode() {
   case "AccessDenied":
    log.Println("Got AccessDenied error, which is the expected result because\n" +
     "the ListBuckets call was made without permissions.")
   default:
    log.Println("Expected AccessDenied, got something else.")
    panic(err)
   }
  }
 } else {
  log.Println("Expected AccessDenied error when calling ListBuckets without
 permissions,\n" +
   "but the call succeeded. Continuing the example anyway...")
 }
 log.Println(strings.Repeat("-", 88))
 return &noPermsConfig
}

// ListBucketsWithAssumedRole performs the following actions:
//
//  1. Creates an AWS Security Token Service (AWS STS) client from the config
 created from
//     the user's access key credentials.
//  2. Gets temporary credentials by assuming the role that grants permission to
 list the
//     buckets.
//  3. Creates an Amazon S3 client from the temporary credentials.
//  4. Lists buckets for the account. Because the temporary credentials are
 generated by
//     assuming the role that grants permission, the action succeeds.
func (scenario AssumeRoleScenario) ListBucketsWithAssumedRole(ctx context.Context,
 noPermsConfig *aws.Config, role *types.Role) {
 log.Println("Let's assume the role that grants permission to list buckets and try
 again.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 stsClient := sts.NewFromConfig(*noPermsConfig)
 tempCredentials, err := stsClient.AssumeRole(ctx, &sts.AssumeRoleInput{
  RoleArn:         role.Arn,
  RoleSessionName: aws.String("AssumeRoleExampleSession"),
  DurationSeconds: aws.Int32(900),
 })
 if err != nil {
  log.Printf("Couldn't assume role %v.\n", *role.RoleName)
```

```
  panic(err)
 }
 log.Printf("Assumed role %v, got temporary credentials.\n", *role.RoleName)
 assumeRoleConfig, err := config.LoadDefaultConfig(ctx,
  config.WithCredentialsProvider(credentials.NewStaticCredentialsProvider(
   *tempCredentials.Credentials.AccessKeyId,
   *tempCredentials.Credentials.SecretAccessKey,
   *tempCredentials.Credentials.SessionToken),
  ),
 )
 if err != nil {
  panic(err)
 }

 // Add test options if this is a test run. This is needed only for testing
 purposes.
 scenario.addTestOptions(&assumeRoleConfig)

 s3Client := s3.NewFromConfig(assumeRoleConfig)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  log.Println("Couldn't list buckets with assumed role credentials.")
  panic(err)
 }
 log.Println("Successfully called ListBuckets with assumed role credentials, \n" +
  "here are some of them:")
 for i := 0; i < len(result.Buckets) && i < 5; i++ {
  log.Printf("\t%v\n", *result.Buckets[i].Name)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup deletes all resources created for the scenario.
func (scenario AssumeRoleScenario) Cleanup(ctx context.Context, user *types.User,
 role *types.Role) {
 if scenario.questioner.AskBool(
  "Do you want to delete the resources created for this example? (y/n)", "y",
 ) {
  policies, err := scenario.roleWrapper.ListAttachedRolePolicies(ctx,
 *role.RoleName)
  if err != nil {
   panic(err)
  }
  for _, policy := range policies {
```

```
  err = scenario.roleWrapper.DetachRolePolicy(ctx, *role.RoleName,
*policy.PolicyArn)
  if err != nil {
   panic(err)
  }
  err = scenario.policyWrapper.DeletePolicy(ctx, *policy.PolicyArn)
  if err != nil {
   panic(err)
  }
  log.Printf("Detached policy %v from role %v and deleted the policy.\n",
   *policy.PolicyName, *role.RoleName)
 }
 err = scenario.roleWrapper.DeleteRole(ctx, *role.RoleName)
 if err != nil {
  panic(err)
 }
 log.Printf("Deleted role %v.\n", *role.RoleName)

 userPols, err := scenario.userWrapper.ListUserPolicies(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 for _, userPol := range userPols {
  err = scenario.userWrapper.DeleteUserPolicy(ctx, *user.UserName, userPol)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted policy %v from user %v.\n", userPol, *user.UserName)
 }
 keys, err := scenario.userWrapper.ListAccessKeys(ctx, *user.UserName)
 if err != nil {
  panic(err)
 }
 for _, key := range keys {
  err = scenario.userWrapper.DeleteAccessKey(ctx, *user.UserName, *key.AccessKeyId)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleted access key %v from user %v.\n", *key.AccessKeyId,
*user.UserName)
 }
 err = scenario.userWrapper.DeleteUser(ctx, *user.UserName)
 if err != nil {
  panic(err)
```

```
  }
  log.Printf("Deleted user %v.\n", *user.UserName)
  log.Println(strings.Repeat("-", 88))
 }

}

// IScenarioHelper abstracts input and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
 Pause(secs int)
}

const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper *ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}
```

Define a struct that wraps account actions.

```
import (
 "context"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)
```

```
// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

Define a struct that wraps policy actions.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
```

```
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
```

```
 }
 return policy, err
}



// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

Define a struct that wraps role actions.

```
import (
 "context"
 "encoding/json"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
```

```
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
```

```
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context, serviceName
 string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}



// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn string,
 roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
```

```
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName, err)
 }
 return err
}



// ListRolePolicies lists the inline policies for a role.
```

```
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListRolePolicies(ctx, &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

Define a struct that wraps user actions.

```
import (
 "context"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
```

```
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
  users = result.Users
 }
 return users, err
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
```

```
   }
  }
 } else {
  user = result.User
 }
 return user, err
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}



// CreateUserPolicy adds an inline policy to a user. This example creates a policy
 that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
```

```
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx, &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName, err)
```

```
 }
 return err
}



// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName string)
 (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
```

```
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
 }
 return err
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - AttachRolePolicy
  - CreateAccessKey
  - CreatePolicy
  - CreateRole
  - CreateUser
  - DeleteAccessKey
  - DeletePolicy
  - DeleteRole
  - DeleteUser
  - DeleteUserPolicy
  - DetachRolePolicy

- PutUserPolicy

Actions

`AttachRolePolicy`

The following code example shows how to use `AttachRolePolicy`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// AttachRolePolicy attaches a policy to a role.
func (wrapper RoleWrapper) AttachRolePolicy(ctx context.Context, policyArn string,
 roleName string) error {
 _, err := wrapper.IamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
```

```
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't attach policy %v to role %v. Here's why: %v\n", policyArn,
 roleName, err)
 }
 return err
}
```

- For API details, see AttachRolePolicy in *AWS SDK for Go API Reference*.

`CreateAccessKey`

The following code example shows how to use `CreateAccessKey`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
```

```
}



// CreateAccessKeyPair creates an access key for a user. The returned access key
 contains
// the ID and secret credentials needed to use the key.
func (wrapper UserWrapper) CreateAccessKeyPair(ctx context.Context, userName string)
 (*types.AccessKey, error) {
 var key *types.AccessKey
 result, err := wrapper.IamClient.CreateAccessKey(ctx, &iam.CreateAccessKeyInput{
  UserName: aws.String(userName)})
 if err != nil {
  log.Printf("Couldn't create access key pair for user %v. Here's why: %v\n",
 userName, err)
 } else {
  key = result.AccessKey
 }
 return key, err
}
```

- For API details, see CreateAccessKey in *AWS SDK for Go API Reference*.

`CreatePolicy`

The following code example shows how to use `CreatePolicy`.

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"
 "encoding/json"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    []string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
}

// CreatePolicy creates a policy that grants a list of actions to the specified
 resource.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper PolicyWrapper) CreatePolicy(ctx context.Context, policyName string,
 actions []string,
 resourceArn string) (*types.Policy, error) {
 var policy *types.Policy
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
```

```
   Resource: aws.String(resourceArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n",
 resourceArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreatePolicy(ctx, &iam.CreatePolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
 })
 if err != nil {
  log.Printf("Couldn't create policy %v. Here's why: %v\n", policyName, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- For API details, see CreatePolicy in *AWS SDK for Go API Reference*.

`CreateRole`

The following code example shows how to use `CreateRole`.

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"
 "encoding/json"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// CreateRole creates a role that trusts a specified user. The trusted user can
 assume
// the role to acquire its permissions.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper RoleWrapper) CreateRole(ctx context.Context, roleName string,
 trustedUserArn string) (*types.Role, error) {
 var role *types.Role
 trustPolicy := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Principal: map[string]string{"AWS": trustedUserArn},
   Action:    []string{"sts:AssumeRole"},
  }},
 }
 policyBytes, err := json.Marshal(trustPolicy)
 if err != nil {
  log.Printf("Couldn't create trust policy for %v. Here's why: %v\n",
 trustedUserArn, err)
  return nil, err
 }
 result, err := wrapper.IamClient.CreateRole(ctx, &iam.CreateRoleInput{
  AssumeRolePolicyDocument: aws.String(string(policyBytes)),
  RoleName:                 aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't create role %v. Here's why: %v\n", roleName, err)
 } else {
```

```
  role = result.Role
 }
 return role, err
}
```

- For API details, see CreateRole in *AWS SDK for Go API Reference*.

`CreateServiceLinkedRole`

The following code example shows how to use `CreateServiceLinkedRole`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}
```

```
// CreateServiceLinkedRole creates a service-linked role that is owned by the
 specified service.
func (wrapper RoleWrapper) CreateServiceLinkedRole(ctx context.Context, serviceName
 string, description string) (
 *types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.CreateServiceLinkedRole(ctx,
 &iam.CreateServiceLinkedRoleInput{
  AWSServiceName: aws.String(serviceName),
  Description:    aws.String(description),
 })
 if err != nil {
  log.Printf("Couldn't create service-linked role %v. Here's why: %v\n",
 serviceName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- For API details, see CreateServiceLinkedRole in *AWS SDK for Go API Reference*.

`CreateUser`

The following code example shows how to use `CreateUser`.

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"
 "encoding/json"
 "errors"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// CreateUser creates a new user with the specified name.
func (wrapper UserWrapper) CreateUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.CreateUser(ctx, &iam.CreateUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create user %v. Here's why: %v\n", userName, err)
 } else {
  user = result.User
 }
 return user, err
}
```

- For API details, see CreateUser in *AWS SDK for Go API Reference*.

`DeleteAccessKey`

The following code example shows how to use `DeleteAccessKey`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteAccessKey deletes an access key from a user.
func (wrapper UserWrapper) DeleteAccessKey(ctx context.Context, userName string,
 keyId string) error {
 _, err := wrapper.IamClient.DeleteAccessKey(ctx, &iam.DeleteAccessKeyInput{
  AccessKeyId: aws.String(keyId),
  UserName:    aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete access key %v. Here's why: %v\n", keyId, err)
 }
 return err
}
```

- For API details, see DeleteAccessKey in *AWS SDK for Go API Reference*.

`DeletePolicy`

The following code example shows how to use `DeletePolicy`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// DeletePolicy deletes a policy.
func (wrapper PolicyWrapper) DeletePolicy(ctx context.Context, policyArn string)
 error {
 _, err := wrapper.IamClient.DeletePolicy(ctx, &iam.DeletePolicyInput{
  PolicyArn: aws.String(policyArn),
```

```
 })
 if err != nil {
  log.Printf("Couldn't delete policy %v. Here's why: %v\n", policyArn, err)
 }
 return err
}
```

- For API details, see DeletePolicy in *AWS SDK for Go API Reference*.

`DeleteRole`

The following code example shows how to use `DeleteRole`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}
```

```
// DeleteRole deletes a role. All attached policies must be detached before a
// role can be deleted.
func (wrapper RoleWrapper) DeleteRole(ctx context.Context, roleName string) error {
 _, err := wrapper.IamClient.DeleteRole(ctx, &iam.DeleteRoleInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't delete role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

- For API details, see DeleteRole in *AWS SDK for Go API Reference*.

`DeleteServiceLinkedRole`

The following code example shows how to use `DeleteServiceLinkedRole`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
```

```
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DeleteServiceLinkedRole deletes a service-linked role.
func (wrapper RoleWrapper) DeleteServiceLinkedRole(ctx context.Context, roleName
 string) error {
 _, err := wrapper.IamClient.DeleteServiceLinkedRole(ctx,
 &iam.DeleteServiceLinkedRoleInput{
  RoleName: aws.String(roleName)},
 )
 if err != nil {
  log.Printf("Couldn't delete service-linked role %v. Here's why: %v\n", roleName,
 err)
 }
 return err
}
```

- For API details, see DeleteServiceLinkedRole in *AWS SDK for Go API Reference*.

`DeleteUser`

The following code example shows how to use `DeleteUser`.

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"
 "encoding/json"
 "errors"
```

```
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteUser deletes a user.
func (wrapper UserWrapper) DeleteUser(ctx context.Context, userName string) error {
 _, err := wrapper.IamClient.DeleteUser(ctx, &iam.DeleteUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- For API details, see DeleteUser in *AWS SDK for Go API Reference*.

`DeleteUserPolicy`

The following code example shows how to use `DeleteUserPolicy`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// DeleteUserPolicy deletes an inline policy from a user.
func (wrapper UserWrapper) DeleteUserPolicy(ctx context.Context, userName string,
 policyName string) error {
 _, err := wrapper.IamClient.DeleteUserPolicy(ctx, &iam.DeleteUserPolicyInput{
  PolicyName: aws.String(policyName),
  UserName:   aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't delete policy from user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- For API details, see DeleteUserPolicy in *AWS SDK for Go API Reference*.

`DetachRolePolicy`

The following code example shows how to use `DetachRolePolicy`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// DetachRolePolicy detaches a policy from a role.
func (wrapper RoleWrapper) DetachRolePolicy(ctx context.Context, roleName string,
 policyArn string) error {
 _, err := wrapper.IamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
  PolicyArn: aws.String(policyArn),
  RoleName:  aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't detach policy from role %v. Here's why: %v\n", roleName, err)
 }
 return err
}
```

- For API details, see DetachRolePolicy in *AWS SDK for Go API Reference*.

`GetAccountPasswordPolicy`

The following code example shows how to use `GetAccountPasswordPolicy`.

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"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// GetAccountPasswordPolicy gets the account password policy for the current
 account.
// If no policy has been set, a NoSuchEntityException is error is returned.
func (wrapper AccountWrapper) GetAccountPasswordPolicy(ctx context.Context)
 (*types.PasswordPolicy, error) {
 var pwPolicy *types.PasswordPolicy
 result, err := wrapper.IamClient.GetAccountPasswordPolicy(ctx,
  &iam.GetAccountPasswordPolicyInput{})
 if err != nil {
```

```
  log.Printf("Couldn't get account password policy. Here's why: %v\n", err)
 } else {
  pwPolicy = result.PasswordPolicy
 }
 return pwPolicy, err
}
```

- For API details, see GetAccountPasswordPolicy in *AWS SDK for Go API Reference*.

`GetPolicy`

The following code example shows how to use `GetPolicy`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}
```

```
// GetPolicy gets data about a policy.
func (wrapper PolicyWrapper) GetPolicy(ctx context.Context, policyArn string)
 (*types.Policy, error) {
 var policy *types.Policy
 result, err := wrapper.IamClient.GetPolicy(ctx, &iam.GetPolicyInput{
  PolicyArn: aws.String(policyArn),
 })
 if err != nil {
  log.Printf("Couldn't get policy %v. Here's why: %v\n", policyArn, err)
 } else {
  policy = result.Policy
 }
 return policy, err
}
```

- For API details, see GetPolicy in *AWS SDK for Go API Reference*.

`GetRole`

The following code example shows how to use `GetRole`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
```

```
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// GetRole gets data about a role.
func (wrapper RoleWrapper) GetRole(ctx context.Context, roleName string)
 (*types.Role, error) {
 var role *types.Role
 result, err := wrapper.IamClient.GetRole(ctx,
  &iam.GetRoleInput{RoleName: aws.String(roleName)})
 if err != nil {
  log.Printf("Couldn't get role %v. Here's why: %v\n", roleName, err)
 } else {
  role = result.Role
 }
 return role, err
}
```

- For API details, see GetRole in *AWS SDK for Go API Reference*.

`GetUser`

The following code example shows how to use `GetUser`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// GetUser gets data about a user.
func (wrapper UserWrapper) GetUser(ctx context.Context, userName string)
 (*types.User, error) {
 var user *types.User
 result, err := wrapper.IamClient.GetUser(ctx, &iam.GetUserInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NoSuchEntityException:
    log.Printf("User %v does not exist.\n", userName)
    err = nil
   default:
    log.Printf("Couldn't get user %v. Here's why: %v\n", userName, err)
   }
  }
 } else {
  user = result.User
 }
 return user, err
}
```

- For API details, see GetUser in *AWS SDK for Go API Reference*.

`ListAccessKeys`

The following code example shows how to use `ListAccessKeys`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListAccessKeys lists the access keys for the specified user.
func (wrapper UserWrapper) ListAccessKeys(ctx context.Context, userName string)
 ([]types.AccessKeyMetadata, error) {
 var keys []types.AccessKeyMetadata
```

```
 result, err := wrapper.IamClient.ListAccessKeys(ctx, &iam.ListAccessKeysInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list access keys for user %v. Here's why: %v\n", userName,
 err)
 } else {
  keys = result.AccessKeyMetadata
 }
 return keys, err
}
```

- For API details, see ListAccessKeys in *AWS SDK for Go API Reference*.

`ListAttachedRolePolicies`

The following code example shows how to use `ListAttachedRolePolicies`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
```

```
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListAttachedRolePolicies lists the policies that are attached to the specified
 role.
func (wrapper RoleWrapper) ListAttachedRolePolicies(ctx context.Context, roleName
 string) ([]types.AttachedPolicy, error) {
 var policies []types.AttachedPolicy
 result, err := wrapper.IamClient.ListAttachedRolePolicies(ctx,
 &iam.ListAttachedRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list attached policies for role %v. Here's why: %v\n",
 roleName, err)
 } else {
  policies = result.AttachedPolicies
 }
 return policies, err
}
```

- For API details, see ListAttachedRolePolicies in *AWS SDK for Go API Reference*.

`ListGroups`

The following code example shows how to use `ListGroups`.

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"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// GroupWrapper encapsulates AWS Identity and Access Management (IAM) group actions
// used in the examples.
// It contains an IAM service client that is used to perform group actions.
type GroupWrapper struct {
 IamClient *iam.Client
}



// ListGroups lists up to maxGroups number of groups.
func (wrapper GroupWrapper) ListGroups(ctx context.Context, maxGroups int32)
 ([]types.Group, error) {
 var groups []types.Group
 result, err := wrapper.IamClient.ListGroups(ctx, &iam.ListGroupsInput{
  MaxItems: aws.Int32(maxGroups),
 })
 if err != nil {
  log.Printf("Couldn't list groups. Here's why: %v\n", err)
 } else {
  groups = result.Groups
 }
 return groups, err
}
```

- For API details, see ListGroups in *AWS SDK for Go API Reference*.

`ListPolicies`

The following code example shows how to use `ListPolicies`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// PolicyWrapper encapsulates AWS Identity and Access Management (IAM) policy
 actions
// used in the examples.
// It contains an IAM service client that is used to perform policy actions.
type PolicyWrapper struct {
 IamClient *iam.Client
}



// ListPolicies gets up to maxPolicies policies.
func (wrapper PolicyWrapper) ListPolicies(ctx context.Context, maxPolicies int32)
 ([]types.Policy, error) {
 var policies []types.Policy
 result, err := wrapper.IamClient.ListPolicies(ctx, &iam.ListPoliciesInput{
  MaxItems: aws.Int32(maxPolicies),
 })
 if err != nil {
  log.Printf("Couldn't list policies. Here's why: %v\n", err)
 } else {
  policies = result.Policies
 }
 return policies, err
```

```
}
```

- For API details, see ListPolicies in *AWS SDK for Go API Reference*.

`ListRolePolicies`

The following code example shows how to use `ListRolePolicies`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
 IamClient *iam.Client
}



// ListRolePolicies lists the inline policies for a role.
func (wrapper RoleWrapper) ListRolePolicies(ctx context.Context, roleName string)
 ([]string, error) {
 var policies []string
```

```
 result, err := wrapper.IamClient.ListRolePolicies(ctx, &iam.ListRolePoliciesInput{
  RoleName: aws.String(roleName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for role %v. Here's why: %v\n", roleName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}
```

- For API details, see ListRolePolicies in *AWS SDK for Go API Reference*.

`ListRoles`

The following code example shows how to use `ListRoles`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// RoleWrapper encapsulates AWS Identity and Access Management (IAM) role actions
// used in the examples.
// It contains an IAM service client that is used to perform role actions.
type RoleWrapper struct {
```

```
 IamClient *iam.Client
}



// ListRoles gets up to maxRoles roles.
func (wrapper RoleWrapper) ListRoles(ctx context.Context, maxRoles int32)
 ([]types.Role, error) {
 var roles []types.Role
 result, err := wrapper.IamClient.ListRoles(ctx,
  &iam.ListRolesInput{MaxItems: aws.Int32(maxRoles)},
 )
 if err != nil {
  log.Printf("Couldn't list roles. Here's why: %v\n", err)
 } else {
  roles = result.Roles
 }
 return roles, err
}
```

- For API details, see ListRoles in *AWS SDK for Go API Reference*.

`ListSAMLProviders`

The following code example shows how to use `ListSAMLProviders`.

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"
 "log"

 "github.com/aws/aws-sdk-go-v2/service/iam"
```

```
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
)

// AccountWrapper encapsulates AWS Identity and Access Management (IAM) account
 actions
// used in the examples.
// It contains an IAM service client that is used to perform account actions.
type AccountWrapper struct {
 IamClient *iam.Client
}



// ListSAMLProviders gets the SAML providers for the account.
func (wrapper AccountWrapper) ListSAMLProviders(ctx context.Context)
 ([]types.SAMLProviderListEntry, error) {
 var providers []types.SAMLProviderListEntry
 result, err := wrapper.IamClient.ListSAMLProviders(ctx,
 &iam.ListSAMLProvidersInput{})
 if err != nil {
  log.Printf("Couldn't list SAML providers. Here's why: %v\n", err)
 } else {
  providers = result.SAMLProviderList
 }
 return providers, err
}
```

- For API details, see ListSAMLProviders in *AWS SDK for Go API Reference*.

`ListUserPolicies`

The following code example shows how to use `ListUserPolicies`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUserPolicies lists the inline policies for the specified user.
func (wrapper UserWrapper) ListUserPolicies(ctx context.Context, userName string)
 ([]string, error) {
 var policies []string
 result, err := wrapper.IamClient.ListUserPolicies(ctx, &iam.ListUserPoliciesInput{
  UserName: aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't list policies for user %v. Here's why: %v\n", userName, err)
 } else {
  policies = result.PolicyNames
 }
 return policies, err
}
```

- For API details, see ListUserPolicies in *AWS SDK for Go API Reference*.

`ListUsers`

The following code example shows how to use `ListUsers`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}



// ListUsers gets up to maxUsers number of users.
func (wrapper UserWrapper) ListUsers(ctx context.Context, maxUsers int32)
 ([]types.User, error) {
 var users []types.User
 result, err := wrapper.IamClient.ListUsers(ctx, &iam.ListUsersInput{
  MaxItems: aws.Int32(maxUsers),
 })
 if err != nil {
  log.Printf("Couldn't list users. Here's why: %v\n", err)
 } else {
```

```
  users = result.Users
 }
 return users, err
}
```

- For API details, see ListUsers in *AWS SDK for Go API Reference*.

`PutUserPolicy`

The following code example shows how to use `PutUserPolicy`.

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"
 "encoding/json"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
 "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/smithy-go"
)

// UserWrapper encapsulates user actions used in the examples.
// It contains an IAM service client that is used to perform user actions.
type UserWrapper struct {
 IamClient *iam.Client
}
```

```
// CreateUserPolicy adds an inline policy to a user. This example creates a policy
 that
// grants a list of actions on a specified role.
// PolicyDocument shows how to work with a policy document as a data structure and
// serialize it to JSON by using Go's JSON marshaler.
func (wrapper UserWrapper) CreateUserPolicy(ctx context.Context, userName string,
 policyName string, actions []string,
 roleArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:   "Allow",
   Action:   actions,
   Resource: aws.String(roleArn),
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document for %v. Here's why: %v\n", roleArn,
 err)
  return err
 }
 _, err = wrapper.IamClient.PutUserPolicy(ctx, &iam.PutUserPolicyInput{
  PolicyDocument: aws.String(string(policyBytes)),
  PolicyName:     aws.String(policyName),
  UserName:       aws.String(userName),
 })
 if err != nil {
  log.Printf("Couldn't create policy for user %v. Here's why: %v\n", userName, err)
 }
 return err
}
```

- For API details, see PutUserPolicy in *AWS SDK for Go API Reference*.

Kinesis examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Kinesis.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Serverless examples

Serverless examples

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Kinesis event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }
```

```
 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting Kinesis batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
```

```
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Lambda examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Lambda.

Basics are code examples that show you how to perform the essential operations within a service.

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.

AWS community contributions are examples that were created and are maintained by multiple teams across AWS. To provide feedback, use the mechanism provided in the linked repositories.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions
- Scenarios
- Serverless examples
- AWS community contributions

Get started

Hello Lambda

The following code example shows how to get started using Lambda.

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"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
)

// main uses the AWS SDK for Go (v2) to create an AWS Lambda client and list up to
 10
// functions 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
 }
 lambdaClient := lambda.NewFromConfig(sdkConfig)

 maxItems := 10
 fmt.Printf("Let's list up to %v functions for your account.\n", maxItems)
 result, err := lambdaClient.ListFunctions(ctx, &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 if err != nil {
  fmt.Printf("Couldn't list functions for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Functions) == 0 {
  fmt.Println("You don't have any functions!")
 } else {
  for _, function := range result.Functions {
   fmt.Printf("\t%v\n", *function.FunctionName)
  }
 }
}
```

- For API details, see ListFunctions in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to:

- Create an IAM role and Lambda function, then upload handler code.
- Invoke the function with a single parameter and get results.
- Update the function code and configure with an environment variable.
- Invoke the function with new parameters and get results. Display the returned execution log.
- List the functions for your account, then clean up resources.

For more information, see Create a Lambda function with the console.

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.

Create an interactive scenario that shows you how to get started with Lambda functions.

```
import (
 "archive/zip"
 "bytes"
 "context"
 "encoding/base64"
 "encoding/json"
 "errors"
 "fmt"
 "log"
 "os"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/iam"
```

```
 iamtypes "github.com/aws/aws-sdk-go-v2/service/iam/types"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/lambda/actions"
)

// GetStartedFunctionsScenario shows you how to use AWS Lambda to perform the
 following
// actions:
//
//  1. Create an AWS Identity and Access Management (IAM) role and Lambda function,
 then upload handler code.
//  2. Invoke the function with a single parameter and get results.
//  3. Update the function code and configure with an environment variable.
//  4. Invoke the function with new parameters and get results. Display the returned
 execution log.
//  5. List the functions for your account, then clean up resources.
type GetStartedFunctionsScenario struct {
 sdkConfig       aws.Config
 functionWrapper actions.FunctionWrapper
 questioner      demotools.IQuestioner
 helper          IScenarioHelper
 isTestRun       bool
}

// NewGetStartedFunctionsScenario constructs a GetStartedFunctionsScenario instance
 from a configuration.
// It uses the specified config to get a Lambda client and create wrappers for the
 actions
// used in the scenario.
func NewGetStartedFunctionsScenario(sdkConfig aws.Config, questioner
 demotools.IQuestioner,
 helper IScenarioHelper) GetStartedFunctionsScenario {
 lambdaClient := lambda.NewFromConfig(sdkConfig)
 return GetStartedFunctionsScenario{
  sdkConfig:       sdkConfig,
  functionWrapper: actions.FunctionWrapper{LambdaClient: lambdaClient},
  questioner:      questioner,
  helper:          helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedFunctionsScenario) Run(ctx context.Context) {
```

```
 defer func() {
  if r := recover(); r != nil {
   log.Printf("Something went wrong with the demo.\n")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the AWS Lambda get started with functions demo.")
 log.Println(strings.Repeat("-", 88))

 role := scenario.GetOrCreateRole(ctx)
 funcName := scenario.CreateFunction(ctx, role)
 scenario.InvokeIncrement(ctx, funcName)
 scenario.UpdateFunction(ctx, funcName)
 scenario.InvokeCalculator(ctx, funcName)
 scenario.ListFunctions(ctx)
 scenario.Cleanup(ctx, role, funcName)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// GetOrCreateRole checks whether the specified role exists and returns it if it
 does.
// Otherwise, a role is created that specifies Lambda as a trusted principal.
// The AWSLambdaBasicExecutionRole managed policy is attached to the role and the
 role
// is returned.
func (scenario GetStartedFunctionsScenario) GetOrCreateRole(ctx context.Context)
 *iamtypes.Role {
 var role *iamtypes.Role
 iamClient := iam.NewFromConfig(scenario.sdkConfig)
 log.Println("First, we need an IAM role that Lambda can assume.")
 roleName := scenario.questioner.Ask("Enter a name for the role:",
 demotools.NotEmpty{})
 getOutput, err := iamClient.GetRole(ctx, &iam.GetRoleInput{
  RoleName: aws.String(roleName)})
 if err != nil {
  var noSuch *iamtypes.NoSuchEntityException
  if errors.As(err, &noSuch) {
   log.Printf("Role %v doesn't exist. Creating it....\n", roleName)
  } else {
   log.Panicf("Couldn't check whether role %v exists. Here's why: %v\n",
```

```
    roleName, err)
  }
 } else {
  role = getOutput.Role
  log.Printf("Found role %v.\n", *role.RoleName)
 }
 if role == nil {
  trustPolicy := PolicyDocument{
   Version: "2012-10-17",
   Statement: []PolicyStatement{{
    Effect:    "Allow",
    Principal: map[string]string{"Service": "lambda.amazonaws.com"},
    Action:    []string{"sts:AssumeRole"},
   }},
  }
  policyArn := "arn:aws:iam::aws:policy/service-role/AWSLambdaBasicExecutionRole"
  createOutput, err := iamClient.CreateRole(ctx, &iam.CreateRoleInput{
   AssumeRolePolicyDocument: aws.String(trustPolicy.String()),
   RoleName:                 aws.String(roleName),
  })
  if err != nil {
   log.Panicf("Couldn't create role %v. Here's why: %v\n", roleName, err)
  }
  role = createOutput.Role
  _, err = iamClient.AttachRolePolicy(ctx, &iam.AttachRolePolicyInput{
   PolicyArn: aws.String(policyArn),
   RoleName:  aws.String(roleName),
  })
  if err != nil {
   log.Panicf("Couldn't attach a policy to role %v. Here's why: %v\n", roleName,
 err)
  }
  log.Printf("Created role %v.\n", *role.RoleName)
  log.Println("Let's give AWS a few seconds to propagate resources...")
  scenario.helper.Pause(10)
 }
 log.Println(strings.Repeat("-", 88))
 return role
}

// CreateFunction creates a Lambda function and uploads a handler written in Python.
// The code for the Python handler is packaged as a []byte in .zip format.
func (scenario GetStartedFunctionsScenario) CreateFunction(ctx context.Context, role
 *iamtypes.Role) string {
```

```
 log.Println("Let's create a function that increments a number.\n" +
  "The function uses the 'lambda_handler_basic.py' script found in the \n" +
  "'handlers' directory of this project.")
 funcName := scenario.questioner.Ask("Enter a name for the Lambda function:",
 demotools.NotEmpty{})
 zipPackage := scenario.helper.CreateDeploymentPackage("lambda_handler_basic.py",
 fmt.Sprintf("%v.py", funcName))
 log.Printf("Creating function %v and waiting for it to be ready.", funcName)
 funcState := scenario.functionWrapper.CreateFunction(ctx, funcName,
 fmt.Sprintf("%v.lambda_handler", funcName),
  role.Arn, zipPackage)
 log.Printf("Your function is %v.", funcState)
 log.Println(strings.Repeat("-", 88))
 return funcName
}

// InvokeIncrement invokes a Lambda function that increments a number. The function
// parameters are contained in a Go struct that is used to serialize the parameters
 to
// a JSON payload that is passed to the function.
// The result payload is deserialized into a Go struct that contains an int value.
func (scenario GetStartedFunctionsScenario) InvokeIncrement(ctx context.Context,
 funcName string) {
 parameters := actions.IncrementParameters{Action: "increment"}
 log.Println("Let's invoke our function. This function increments a number.")
 parameters.Number = scenario.questioner.AskInt("Enter a number to increment:",
 demotools.NotEmpty{})
 log.Printf("Invoking %v with %v...\n", funcName, parameters.Number)
 invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, parameters, false)
 var payload actions.LambdaResultInt
 err := json.Unmarshal(invokeOutput.Payload, &payload)
 if err != nil {
  log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
   funcName, err)
 }
 log.Printf("Invoking %v with %v returned %v.\n", funcName, parameters.Number,
 payload)
 log.Println(strings.Repeat("-", 88))
}

// UpdateFunction updates the code for a Lambda function by uploading a simple
 arithmetic
// calculator written in Python. The code for the Python handler is packaged as a
// []byte in .zip format.
```

```
// After the code is updated, the configuration is also updated with a new log
// level that instructs the handler to log additional information.
func (scenario GetStartedFunctionsScenario) UpdateFunction(ctx context.Context,
 funcName string) {
 log.Println("Let's update the function to an arithmetic calculator.\n" +
  "The function uses the 'lambda_handler_calculator.py' script found in the \n" +
  "'handlers' directory of this project.")
 scenario.questioner.Ask("Press Enter when you're ready.")
 log.Println("Creating deployment package...")
 zipPackage :=
 scenario.helper.CreateDeploymentPackage("lambda_handler_calculator.py",
  fmt.Sprintf("%v.py", funcName))
 log.Println("...and updating the Lambda function and waiting for it to be ready.")
 funcState := scenario.functionWrapper.UpdateFunctionCode(ctx, funcName, zipPackage)
 log.Printf("Updated function %v. Its current state is %v.", funcName, funcState)
 log.Println("This function uses an environment variable to control logging level.")
 log.Println("Let's set it to DEBUG to get the most logging.")
 scenario.functionWrapper.UpdateFunctionConfiguration(ctx, funcName,
  map[string]string{"LOG_LEVEL": "DEBUG"})
 log.Println(strings.Repeat("-", 88))
}

// InvokeCalculator invokes the Lambda calculator function. The parameters are
 stored in a
// Go struct that is used to serialize the parameters to a JSON payload. That
 payload is then passed
// to the function.
// The result payload is deserialized to a Go struct that stores the result as
 either an
// int or float32, depending on the kind of operation that was specified.
func (scenario GetStartedFunctionsScenario) InvokeCalculator(ctx context.Context,
 funcName string) {
 wantInvoke := true
 choices := []string{"plus", "minus", "times", "divided-by"}
 for wantInvoke {
  choice := scenario.questioner.AskChoice("Select an arithmetic operation:\n",
 choices)
  x := scenario.questioner.AskInt("Enter a value for x:", demotools.NotEmpty{})
  y := scenario.questioner.AskInt("Enter a value for y:", demotools.NotEmpty{})
  log.Printf("Invoking %v %v %v...", x, choices[choice], y)
  calcParameters := actions.CalculatorParameters{
   Action: choices[choice],
   X:      x,
   Y:      y,
```

```
  }
  invokeOutput := scenario.functionWrapper.Invoke(ctx, funcName, calcParameters,
 true)
  var payload any
  if choice == 3 { // divide-by results in a float.
   payload = actions.LambdaResultFloat{}
  } else {
   payload = actions.LambdaResultInt{}
  }
  err := json.Unmarshal(invokeOutput.Payload, &payload)
  if err != nil {
   log.Panicf("Couldn't unmarshal payload from invoking %v. Here's why: %v\n",
    funcName, err)
  }
  log.Printf("Invoking %v with %v %v %v returned %v.\n", funcName,
   calcParameters.X, calcParameters.Action, calcParameters.Y, payload)
  scenario.questioner.Ask("Press Enter to see the logs from the call.")
  logRes, err := base64.StdEncoding.DecodeString(*invokeOutput.LogResult)
  if err != nil {
   log.Panicf("Couldn't decode log result. Here's why: %v\n", err)
  }
  log.Println(string(logRes))
  wantInvoke = scenario.questioner.AskBool("Do you want to calculate again? (y/n)",
 "y")
 }
 log.Println(strings.Repeat("-", 88))
}

// ListFunctions lists up to the specified number of functions for your account.
func (scenario GetStartedFunctionsScenario) ListFunctions(ctx context.Context) {
 count := scenario.questioner.AskInt(
  "Let's list functions for your account. How many do you want to see?",
 demotools.NotEmpty{})
 functions := scenario.functionWrapper.ListFunctions(ctx, count)
 log.Printf("Found %v functions:", len(functions))
 for _, function := range functions {
  log.Printf("\t%v", *function.FunctionName)
 }
 log.Println(strings.Repeat("-", 88))
}

// Cleanup removes the IAM and Lambda resources created by the example.
func (scenario GetStartedFunctionsScenario) Cleanup(ctx context.Context, role
 *iamtypes.Role, funcName string) {
```

```
 if scenario.questioner.AskBool("Do you want to clean up resources created for this
 example? (y/n)",
  "y") {
  iamClient := iam.NewFromConfig(scenario.sdkConfig)
  policiesOutput, err := iamClient.ListAttachedRolePolicies(ctx,
   &iam.ListAttachedRolePoliciesInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't get policies attached to role %v. Here's why: %v\n",
    *role.RoleName, err)
  }
  for _, policy := range policiesOutput.AttachedPolicies {
   _, err = iamClient.DetachRolePolicy(ctx, &iam.DetachRolePolicyInput{
    PolicyArn: policy.PolicyArn, RoleName: role.RoleName,
   })
   if err != nil {
    log.Panicf("Couldn't detach policy %v from role %v. Here's why: %v\n",
     *policy.PolicyArn, *role.RoleName, err)
   }
  }
  _, err = iamClient.DeleteRole(ctx, &iam.DeleteRoleInput{RoleName: role.RoleName})
  if err != nil {
   log.Panicf("Couldn't delete role %v. Here's why: %v\n", *role.RoleName, err)
  }
  log.Printf("Deleted role %v.\n", *role.RoleName)

  scenario.functionWrapper.DeleteFunction(ctx, funcName)
  log.Printf("Deleted function %v.\n", funcName)
 } else {
  log.Println("Okay. Don't forget to delete the resources when you're done with
 them.")
 }
}

// IScenarioHelper abstracts I/O and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 Pause(secs int)
 CreateDeploymentPackage(sourceFile string, destinationFile string) *bytes.Buffer
}

// ScenarioHelper lets the caller specify the path to Lambda handler functions.
type ScenarioHelper struct {
 HandlerPath string
}
```

```
// Pause waits for the specified number of seconds.
func (helper *ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// CreateDeploymentPackage creates an AWS Lambda deployment package from a source
 file. The
// deployment package is stored in .zip format in a bytes.Buffer. The buffer can be
// used to pass a []byte to Lambda when creating the function.
// The specified destinationFile is the name to give the file when it's deployed to
 Lambda.
func (helper *ScenarioHelper) CreateDeploymentPackage(sourceFile string,
 destinationFile string) *bytes.Buffer {
 var err error
 buffer := &bytes.Buffer{}
 writer := zip.NewWriter(buffer)
 zFile, err := writer.Create(destinationFile)
 if err != nil {
  log.Panicf("Couldn't create destination archive %v. Here's why: %v\n",
 destinationFile, err)
 }
 sourceBody, err := os.ReadFile(fmt.Sprintf("%v/%v", helper.HandlerPath,
 sourceFile))
 if err != nil {
  log.Panicf("Couldn't read handler source file %v. Here's why: %v\n",
   sourceFile, err)
 } else {
  _, err = zFile.Write(sourceBody)
  if err != nil {
   log.Panicf("Couldn't write handler %v to zip archive. Here's why: %v\n",
    sourceFile, err)
  }
 }
 err = writer.Close()
 if err != nil {
  log.Panicf("Couldn't close zip writer. Here's why: %v\n", err)
 }
 return buffer
}
```

Create a struct that wraps individual Lambda actions.

```
import (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}


// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName string)
 types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx, &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
```

```
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in the
```

```
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context, functionName
 string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n", functionName,
 err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}



// UpdateFunctionConfiguration updates a map of environment variables configured for
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v", functionName,
 err)
 }
}
```

```
// ListFunctions lists up to maxItems functions for the account. This function uses
 a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
```

```
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}



// IncrementParameters is used to serialize parameters to the increment Lambda
 handler.
type IncrementParameters struct {
 Action string `json:"action"`
 Number int    `json:"number"`
}

// CalculatorParameters is used to serialize parameters to the calculator Lambda
 handler.
type CalculatorParameters struct {
 Action string `json:"action"`
 X      int    `json:"x"`
 Y      int    `json:"y"`
}

// LambdaResultInt is used to deserialize an int result from a Lambda handler.
type LambdaResultInt struct {
 Result int `json:"result"`
}

// LambdaResultFloat is used to deserialize a float32 result from a Lambda handler.
type LambdaResultFloat struct {
 Result float32 `json:"result"`
```

```
}
```

Define a Lambda handler that increments a number.

```
import logging

logger = logging.getLogger()
logger.setLevel(logging.INFO)


def lambda_handler(event, context):
    """
    Accepts an action and a single number, performs the specified action on the
 number,
    and returns the result. The only allowable action is 'increment'.

    :param event: The event dict that contains the parameters sent when the function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the action.
    """
    result = None
    action = event.get("action")
    if action == "increment":
        result = event.get("number", 0) + 1
        logger.info("Calculated result of %s", result)
    else:
        logger.error("%s is not a valid action.", action)

    response = {"result": result}
    return response
```

Define a second Lambda handler that performs arithmetic operations.

```
import logging
import os
```

```
logger = logging.getLogger()

# Define a list of Python lambda functions that are called by this AWS Lambda
 function.
ACTIONS = {
    "plus": lambda x, y: x + y,
    "minus": lambda x, y: x - y,
    "times": lambda x, y: x * y,
    "divided-by": lambda x, y: x / y,
}


def lambda_handler(event, context):
    """
    Accepts an action and two numbers, performs the specified action on the numbers,
    and returns the result.

    :param event: The event dict that contains the parameters sent when the function
                  is invoked.
    :param context: The context in which the function is called.
    :return: The result of the specified action.
    """
    # Set the log level based on a variable configured in the Lambda environment.
    logger.setLevel(os.environ.get("LOG_LEVEL", logging.INFO))
    logger.debug("Event: %s", event)

    action = event.get("action")
    func = ACTIONS.get(action)
    x = event.get("x")
    y = event.get("y")
    result = None
    try:
        if func is not None and x is not None and y is not None:
            result = func(x, y)
            logger.info("%s %s %s is %s", x, action, y, result)
        else:
            logger.error("I can't calculate %s %s %s.", x, action, y)
    except ZeroDivisionError:
        logger.warning("I can't divide %s by 0!", x)

    response = {"result": result}
    return response
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateFunction
  - DeleteFunction
  - GetFunction
  - Invoke
  - ListFunctions
  - UpdateFunctionCode
  - UpdateFunctionConfiguration

Actions

`CreateFunction`

The following code example shows how to use `CreateFunction`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)
```

```
// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// CreateFunction creates a new Lambda function from code contained in the
 zipPackage
// buffer. The specified handlerName must match the name of the file and function
// contained in the uploaded code. The role specified by iamRoleArn is assumed by
// Lambda and grants specific permissions.
// When the function already exists, types.StateActive is returned.
// When the function is created, a lambda.FunctionActiveV2Waiter is used to wait
 until the
// function is active.
func (wrapper FunctionWrapper) CreateFunction(ctx context.Context, functionName
 string, handlerName string,
 iamRoleArn *string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.CreateFunction(ctx, &lambda.CreateFunctionInput{
  Code:         &types.FunctionCode{ZipFile: zipPackage.Bytes()},
  FunctionName: aws.String(functionName),
  Role:         iamRoleArn,
  Handler:      aws.String(handlerName),
  Publish:      true,
  Runtime:      types.RuntimePython39,
 })
 if err != nil {
  var resConflict *types.ResourceConflictException
  if errors.As(err, &resConflict) {
   log.Printf("Function %v already exists.\n", functionName)
   state = types.StateActive
  } else {
   log.Panicf("Couldn't create function %v. Here's why: %v\n", functionName, err)
  }
 } else {
  waiter := lambda.NewFunctionActiveV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
```

```
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
 return state
}
```

- For API details, see CreateFunction in *AWS SDK for Go API Reference*.

`DeleteFunction`

The following code example shows how to use `DeleteFunction`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
```

```
 LambdaClient *lambda.Client
}



// DeleteFunction deletes the Lambda function specified by functionName.
func (wrapper FunctionWrapper) DeleteFunction(ctx context.Context, functionName
 string) {
 _, err := wrapper.LambdaClient.DeleteFunction(ctx, &lambda.DeleteFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't delete function %v. Here's why: %v\n", functionName, err)
 }
}
```

- For API details, see DeleteFunction in *AWS SDK for Go API Reference*.

`GetFunction`

The following code example shows how to use `GetFunction`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// GetFunction gets data about the Lambda function specified by functionName.
func (wrapper FunctionWrapper) GetFunction(ctx context.Context, functionName string)
 types.State {
 var state types.State
 funcOutput, err := wrapper.LambdaClient.GetFunction(ctx, &lambda.GetFunctionInput{
  FunctionName: aws.String(functionName),
 })
 if err != nil {
  log.Panicf("Couldn't get function %v. Here's why: %v\n", functionName, err)
 } else {
  state = funcOutput.Configuration.State
 }
 return state
}
```

- For API details, see GetFunction in *AWS SDK for Go API Reference*.

Invoke

The following code example shows how to use `Invoke`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// Invoke invokes the Lambda function specified by functionName, passing the
 parameters
// as a JSON payload. When getLog is true, types.LogTypeTail is specified, which
 tells
// Lambda to include the last few log lines in the returned result.
func (wrapper FunctionWrapper) Invoke(ctx context.Context, functionName string,
 parameters any, getLog bool) *lambda.InvokeOutput {
 logType := types.LogTypeNone
 if getLog {
  logType = types.LogTypeTail
 }
 payload, err := json.Marshal(parameters)
 if err != nil {
  log.Panicf("Couldn't marshal parameters to JSON. Here's why %v\n", err)
 }
 invokeOutput, err := wrapper.LambdaClient.Invoke(ctx, &lambda.InvokeInput{
  FunctionName: aws.String(functionName),
  LogType:      logType,
  Payload:      payload,
 })
 if err != nil {
```

```
  log.Panicf("Couldn't invoke function %v. Here's why: %v\n", functionName, err)
 }
 return invokeOutput
}
```

- For API details, see Invoke in *AWS SDK for Go API Reference*.

`ListFunctions`

The following code example shows how to use `ListFunctions`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}
```

```
// ListFunctions lists up to maxItems functions for the account. This function uses
 a
// lambda.ListFunctionsPaginator to paginate the results.
func (wrapper FunctionWrapper) ListFunctions(ctx context.Context, maxItems int)
 []types.FunctionConfiguration {
 var functions []types.FunctionConfiguration
 paginator := lambda.NewListFunctionsPaginator(wrapper.LambdaClient,
 &lambda.ListFunctionsInput{
  MaxItems: aws.Int32(int32(maxItems)),
 })
 for paginator.HasMorePages() && len(functions) < maxItems {
  pageOutput, err := paginator.NextPage(ctx)
  if err != nil {
   log.Panicf("Couldn't list functions for your account. Here's why: %v\n", err)
  }
  functions = append(functions, pageOutput.Functions...)
 }
 return functions
}
```

- For API details, see ListFunctions in *AWS SDK for Go API Reference*.

`UpdateFunctionCode`

The following code example shows how to use `UpdateFunctionCode`.

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 (
 "bytes"
 "context"
 "encoding/json"
```

```
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionCode updates the code for the Lambda function specified by
 functionName.
// The existing code for the Lambda function is entirely replaced by the code in the
// zipPackage buffer. After the update action is called, a
 lambda.FunctionUpdatedV2Waiter
// is used to wait until the update is successful.
func (wrapper FunctionWrapper) UpdateFunctionCode(ctx context.Context, functionName
 string, zipPackage *bytes.Buffer) types.State {
 var state types.State
 _, err := wrapper.LambdaClient.UpdateFunctionCode(ctx,
 &lambda.UpdateFunctionCodeInput{
  FunctionName: aws.String(functionName), ZipFile: zipPackage.Bytes(),
 })
 if err != nil {
  log.Panicf("Couldn't update code for function %v. Here's why: %v\n", functionName,
 err)
 } else {
  waiter := lambda.NewFunctionUpdatedV2Waiter(wrapper.LambdaClient)
  funcOutput, err := waiter.WaitForOutput(ctx, &lambda.GetFunctionInput{
   FunctionName: aws.String(functionName)}, 1*time.Minute)
  if err != nil {
   log.Panicf("Couldn't wait for function %v to be active. Here's why: %v\n",
 functionName, err)
  } else {
   state = funcOutput.Configuration.State
  }
 }
```

```
 return state
}
```

- For API details, see UpdateFunctionCode in *AWS SDK for Go API Reference*.

`UpdateFunctionConfiguration`

The following code example shows how to use `UpdateFunctionConfiguration`.

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 (
 "bytes"
 "context"
 "encoding/json"
 "errors"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/lambda"
 "github.com/aws/aws-sdk-go-v2/service/lambda/types"
)

// FunctionWrapper encapsulates function actions used in the examples.
// It contains an AWS Lambda service client that is used to perform user actions.
type FunctionWrapper struct {
 LambdaClient *lambda.Client
}



// UpdateFunctionConfiguration updates a map of environment variables configured for
```

```
// the Lambda function specified by functionName.
func (wrapper FunctionWrapper) UpdateFunctionConfiguration(ctx context.Context,
 functionName string, envVars map[string]string) {
 _, err := wrapper.LambdaClient.UpdateFunctionConfiguration(ctx,
 &lambda.UpdateFunctionConfigurationInput{
  FunctionName: aws.String(functionName),
  Environment:  &types.Environment{Variables: envVars},
 })
 if err != nil {
  log.Panicf("Couldn't update configuration for %v. Here's why: %v", functionName,
 err)
 }
}
```

- For API details, see UpdateFunctionConfiguration in *AWS SDK for Go API Reference*.

Scenarios

Automatically confirm known users with a Lambda function

The following code example shows 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.

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

Automatically migrate known users with a Lambda function

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.

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

Write custom activity data with a Lambda function after Amazon Cognito user authentication

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.

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

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
```

```
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()
```

```
 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Invoke a Lambda function from a Kinesis trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a Kinesis stream. The function retrieves the Kinesis payload, decodes from Base64, and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Kinesis event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent) error {
 if len(kinesisEvent.Records) == 0 {
  log.Printf("empty Kinesis event received")
  return nil
 }

 for _, record := range kinesisEvent.Records {
  log.Printf("processed Kinesis event with EventId: %v", record.EventID)
  recordDataBytes := record.Kinesis.Data
  recordDataText := string(recordDataBytes)
  log.Printf("record data: %v", recordDataText)
  // TODO: Do interesting work based on the new data
 }
 log.Printf("successfully processed %v records", len(kinesisEvent.Records))
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Invoke a Lambda function from a DynamoDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DynamoDB stream. The function retrieves the DynamoDB payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a DynamoDB event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-lambda-go/events"
 "fmt"
)

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*string, error)
 {
 if len(event.Records) == 0 {
  return nil, fmt.Errorf("received empty event")
 }

 for _, record := range event.Records {
   LogDynamoDBRecord(record)
 }

 message := fmt.Sprintf("Records processed: %d", len(event.Records))
 return &message, nil
}

func main() {
 lambda.Start(HandleRequest)
}

func LogDynamoDBRecord(record events.DynamoDBEventRecord){
 fmt.Println(record.EventID)
 fmt.Println(record.EventName)
 fmt.Printf("%+v\n", record.Change)
```

```
}
```

Invoke a Lambda function from a Amazon DocumentDB trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from a DocumentDB change stream. The function retrieves the DocumentDB payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming a Amazon DocumentDB event with Lambda using Go.

```
package main

import (
 "context"
 "encoding/json"
 "fmt"

 "github.com/aws/aws-lambda-go/lambda"
)

type Event struct {
 Events []Record `json:"events"`
}

type Record struct {
 Event struct {
  OperationType string `json:"operationType"`
  NS            struct {
   DB   string `json:"db"`
   Coll string `json:"coll"`
  } `json:"ns"`
  FullDocument interface{} `json:"fullDocument"`
```

```
 } `json:"event"`
}

func main() {
 lambda.Start(handler)
}

func handler(ctx context.Context, event Event) (string, error) {
 fmt.Println("Loading function")
 for _, record := range event.Events {
  logDocumentDBEvent(record)
 }

 return "OK", nil
}

func logDocumentDBEvent(record Record) {
 fmt.Printf("Operation type: %s\n", record.Event.OperationType)
 fmt.Printf("db: %s\n", record.Event.NS.DB)
 fmt.Printf("collection: %s\n", record.Event.NS.Coll)
 docBytes, _ := json.MarshalIndent(record.Event.FullDocument, "", "  ")
 fmt.Printf("Full document: %s\n", string(docBytes))
}
```

Invoke a Lambda function from an Amazon MSK trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from an Amazon MSK cluster. The function retrieves the MSK payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an Amazon MSK event with Lambda using Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an S3 event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
```

```
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
```

```
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
```

```
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Reporting batch item failures for Lambda functions with a Kinesis trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a Kinesis stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting Kinesis batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
```

```
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, kinesisEvent events.KinesisEvent)
 (map[string]interface{}, error) {
 batchItemFailures := []map[string]interface{}{}

 for _, record := range kinesisEvent.Records {
  curRecordSequenceNumber := ""

  // Process your record
  if /* Your record processing condition here */ {
   curRecordSequenceNumber = record.Kinesis.SequenceNumber
  }

  // Add a condition to check if the record processing failed
  if curRecordSequenceNumber != "" {
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": curRecordSequenceNumber})
  }
 }

 kinesisBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return kinesisBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Reporting batch item failures for Lambda functions with a DynamoDB trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from a DynamoDB stream. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting DynamoDB batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

type BatchItemFailure struct {
 ItemIdentifier string `json:"ItemIdentifier"`
}

type BatchResult struct {
 BatchItemFailures []BatchItemFailure `json:"BatchItemFailures"`
}

func HandleRequest(ctx context.Context, event events.DynamoDBEvent) (*BatchResult,
 error) {
 var batchItemFailures []BatchItemFailure
 curRecordSequenceNumber := ""

 for _, record := range event.Records {
  // Process your record
  curRecordSequenceNumber = record.Change.SequenceNumber
```

```
 }

 if curRecordSequenceNumber != "" {
  batchItemFailures = append(batchItemFailures, BatchItemFailure{ItemIdentifier:
 curRecordSequenceNumber})
 }

 batchResult := BatchResult{
  BatchItemFailures: batchItemFailures,
 }

 return &batchResult, nil
}

func main() {
 lambda.Start(HandleRequest)
}
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from an SQS queue. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
```

```
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

AWS community contributions

Build and test a serverless application

The following code example shows how to build and test a serverless application using API Gateway with Lambda and DynamoDB

SDK for Go V2

Shows how to build and test a serverless application that consists of an API Gateway with Lambda and DynamoDB using the Go SDK.

For complete source code and instructions on how to set up and run, see the full example on GitHub.

Services used in this example

- API Gateway
- DynamoDB
- Lambda

Amazon MSK examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon MSK.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Serverless examples

Serverless examples

Invoke a Lambda function from an Amazon MSK trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving records from an Amazon MSK cluster. The function retrieves the MSK payload and logs the record contents.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an Amazon MSK event with Lambda using Go.

```
package main

import (
 "encoding/base64"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.KafkaEvent) {
 for key, records := range event.Records {
  fmt.Println("Key:", key)

  for _, record := range records {
   fmt.Println("Record:", record)

   decodedValue, _ := base64.StdEncoding.DecodeString(record.Value)
   message := string(decodedValue)
   fmt.Println("Message:", message)
  }
 }
}

func main() {
 lambda.Start(handler)
}
```

Partner Central examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Partner Central.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Actions

Actions

`GetOpportunity`

The following code example shows how to use `GetOpportunity`.

SDK for Go V2

Get an opportunity.

```
package main

import (
 "context"
 "encoding/json"
 "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/partnercentralselling"
)

func main() {
 config, err := config.LoadDefaultConfig(context.TODO())

 if err != nil {
  log.Fatal(err)
 }

 config.Region = "us-east-1"

 client := partnercentralselling.NewFromConfig(config)

 output, err := client.GetOpportunity(context.TODO(),
 &partnercentralselling.GetOpportunityInput{
  Identifier: aws.String("O1111111"),
  Catalog:    aws.String("AWS"),
```

```
 })

 if err != nil {
  log.Fatal(err)
 }
 log.Println("printing opportuniy...\n")

 jsonOutput, err := json.MarshalIndent(output, "", "    ")

 fmt.Println(string(jsonOutput))
}
```

- For API details, see GetOpportunity in *AWS SDK for Go API Reference*.

`ListOpportunities`

The following code example shows how to use `ListOpportunities`.

SDK for Go V2

List opportunities.

```
package main

import (
 "context"
 "encoding/json"
 "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/partnercentralselling"
)

func main() {
 config, err := config.LoadDefaultConfig(context.TODO())

 if err != nil {
  log.Fatal(err)
 }
```

```
 config.Region = "us-east-1"

 client := partnercentralselling.NewFromConfig(config)

 output, err := client.ListOpportunities(context.TODO(),
 &partnercentralselling.ListOpportunitiesInput{
  MaxResults: aws.Int32(2),
  Catalog:    aws.String("AWS"),
 })

 if err != nil {
  log.Fatal(err)
 }

 jsonOutput, err := json.MarshalIndent(output, "", "    ")
 fmt.Println(string(jsonOutput))
}
```

- For API details, see ListOpportunities in *AWS SDK for Go API Reference*.

Amazon RDS examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon RDS.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions
- Serverless examples

Get started

Hello Amazon RDS

The following code example shows how to get started using Amazon RDS.

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"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/rds"
)

// main uses the AWS SDK for Go V2 to create an Amazon Relational Database Service
 (Amazon RDS)
// client and list up to 20 DB instances 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
 }
 rdsClient := rds.NewFromConfig(sdkConfig)
 const maxInstances = 20
```

```
 fmt.Printf("Let's list up to %v DB instances.\n", maxInstances)
 output, err := rdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{MaxRecords: aws.Int32(maxInstances)})
 if err != nil {
  fmt.Printf("Couldn't list DB instances: %v\n", err)
  return
 }
 if len(output.DBInstances) == 0 {
  fmt.Println("No DB instances found.")
 } else {
  for _, instance := range output.DBInstances {
   fmt.Printf("DB instance %v has database %v.\n", *instance.DBInstanceIdentifier,
    *instance.DBName)
  }
 }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to:

- Create a custom DB parameter group and set parameter values.
- Create a DB instance that's configured to use the parameter group. The DB instance also contains a database.
- Take a snapshot of the instance.
- Delete the instance and parameter group.

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"
 "fmt"
 "log"
 "sort"
 "strconv"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/rds/actions"
 "github.com/google/uuid"
)

// GetStartedInstances is an interactive example that shows you how to use the AWS
 SDK for Go
// with Amazon Relation Database Service (Amazon RDS) to do the following:
//
//  1. Create a custom DB parameter group and set parameter values.
//  2. Create a DB instance that is configured to use the parameter group. The DB
 instance
//     also contains a database.
//  3. Take a snapshot of the DB instance.
//  4. Delete the DB instance and parameter group.
type GetStartedInstances struct {
 sdkConfig  aws.Config
 instances  actions.DbInstances
 questioner demotools.IQuestioner
 helper     IScenarioHelper
 isTestRun  bool
}

// NewGetStartedInstances constructs a GetStartedInstances instance from a
 configuration.
// It uses the specified config to get an Amazon RDS
// client and create wrappers for the actions used in the scenario.
func NewGetStartedInstances(sdkConfig aws.Config, questioner demotools.IQuestioner,
 helper IScenarioHelper) GetStartedInstances {
```

```
 rdsClient := rds.NewFromConfig(sdkConfig)
 return GetStartedInstances{
  sdkConfig:  sdkConfig,
  instances:  actions.DbInstances{RdsClient: rdsClient},
  questioner: questioner,
  helper:     helper,
 }
}

// Run runs the interactive scenario.
func (scenario GetStartedInstances) Run(ctx context.Context, dbEngine string,
 parameterGroupName string,
 instanceName string, dbName string) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon Relational Database Service (Amazon RDS) DB
 Instance demo.")
 log.Println(strings.Repeat("-", 88))

 parameterGroup := scenario.CreateParameterGroup(ctx, dbEngine, parameterGroupName)
 scenario.SetUserParameters(ctx, parameterGroupName)
 instance := scenario.CreateInstance(ctx, instanceName, dbEngine, dbName,
 parameterGroup)
 scenario.DisplayConnection(instance)
 scenario.CreateSnapshot(ctx, instance)
 scenario.Cleanup(ctx, instance, parameterGroup)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}

// CreateParameterGroup shows how to get available engine versions for a specified
// database engine and create a DB parameter group that is compatible with a
// selected engine family.
func (scenario GetStartedInstances) CreateParameterGroup(ctx context.Context,
 dbEngine string,
 parameterGroupName string) *types.DBParameterGroup {
```

```
log.Printf("Checking for an existing DB parameter group named %v.\n",
 parameterGroupName)
parameterGroup, err := scenario.instances.GetParameterGroup(ctx,
parameterGroupName)
if err != nil {
 panic(err)
}
if parameterGroup == nil {
 log.Printf("Getting available database engine versions for %v.\n", dbEngine)
 engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine, "")
 if err != nil {
  panic(err)
 }

 familySet := map[string]struct{}{}
 for _, family := range engineVersions {
  familySet[*family.DBParameterGroupFamily] = struct{}{}
 }
 var families []string
 for family := range familySet {
  families = append(families, family)
 }
 sort.Strings(families)
 familyIndex := scenario.questioner.AskChoice("Which family do you want to use?\n",
families)
 log.Println("Creating a DB parameter group.")
 _, err = scenario.instances.CreateParameterGroup(
  ctx, parameterGroupName, families[familyIndex], "Example parameter group.")
 if err != nil {
  panic(err)
 }
 parameterGroup, err = scenario.instances.GetParameterGroup(ctx,
parameterGroupName)
 if err != nil {
  panic(err)
 }
}
log.Printf("Parameter group %v:\n", *parameterGroup.DBParameterGroupFamily)
log.Printf("\tName: %v\n", *parameterGroup.DBParameterGroupName)
log.Printf("\tARN: %v\n", *parameterGroup.DBParameterGroupArn)
log.Printf("\tFamily: %v\n", *parameterGroup.DBParameterGroupFamily)
log.Printf("\tDescription: %v\n", *parameterGroup.Description)
log.Println(strings.Repeat("-", 88))
return parameterGroup
```

```
}

// SetUserParameters shows how to get the parameters contained in a custom parameter
// group and update some of the parameter values in the group.
func (scenario GetStartedInstances) SetUserParameters(ctx context.Context,
 parameterGroupName string) {
 log.Println("Let's set some parameter values in your parameter group.")
 dbParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName, "")
 if err != nil {
  panic(err)
 }
 var updateParams []types.Parameter
 for _, dbParam := range dbParameters {
  if strings.HasPrefix(*dbParam.ParameterName, "auto_increment") &&
   *dbParam.IsModifiable && *dbParam.DataType == "integer" {
   log.Printf("The %v parameter is described as:\n\t%v",
    *dbParam.ParameterName, *dbParam.Description)
   rangeSplit := strings.Split(*dbParam.AllowedValues, "-")
   lower, _ := strconv.Atoi(rangeSplit[0])
   upper, _ := strconv.Atoi(rangeSplit[1])
   newValue := scenario.questioner.AskInt(
    fmt.Sprintf("Enter a value between %v and %v:", lower, upper),
    demotools.InIntRange{Lower: lower, Upper: upper})
   dbParam.ParameterValue = aws.String(strconv.Itoa(newValue))
   updateParams = append(updateParams, dbParam)
  }
 }
 err = scenario.instances.UpdateParameters(ctx, parameterGroupName, updateParams)
 if err != nil {
  panic(err)
 }
 log.Println("To get a list of parameters that you set previously, specify a source
 of 'user'.")
 userParameters, err := scenario.instances.GetParameters(ctx, parameterGroupName,
 "user")
 if err != nil {
  panic(err)
 }
 log.Println("Here are the parameters you set:")
 for _, param := range userParameters {
  log.Printf("\t%v: %v\n", *param.ParameterName, *param.ParameterValue)
 }
 log.Println(strings.Repeat("-", 88))
}
```

```
// CreateInstance shows how to create a DB instance that contains a database of a
// specified type. The database is also configured to use a custom DB parameter
 group.
func (scenario GetStartedInstances) CreateInstance(ctx context.Context, instanceName
 string, dbEngine string,
 dbName string, parameterGroup *types.DBParameterGroup) *types.DBInstance {

 log.Println("Checking for an existing DB instance.")
 instance, err := scenario.instances.GetInstance(ctx, instanceName)
 if err != nil {
  panic(err)
 }
 if instance == nil {
  adminUsername := scenario.questioner.Ask(
   "Enter an administrator username for the database: ", demotools.NotEmpty{})
  adminPassword := scenario.questioner.AskPassword(
   "Enter a password for the administrator (at least 8 characters): ", 7)
  engineVersions, err := scenario.instances.GetEngineVersions(ctx, dbEngine,
   *parameterGroup.DBParameterGroupFamily)
  if err != nil {
   panic(err)
  }
  var engineChoices []string
  for _, engine := range engineVersions {
   engineChoices = append(engineChoices, *engine.EngineVersion)
  }
  engineIndex := scenario.questioner.AskChoice(
   "The available engines for your parameter group are:\n", engineChoices)
  engineSelection := engineVersions[engineIndex]
  instOpts, err := scenario.instances.GetOrderableInstances(ctx,
 *engineSelection.Engine,
   *engineSelection.EngineVersion)
  if err != nil {
   panic(err)
  }
  optSet := map[string]struct{}{}
  for _, opt := range instOpts {
   if strings.Contains(*opt.DBInstanceClass, "micro") {
    optSet[*opt.DBInstanceClass] = struct{}{}
   }
  }
  var optChoices []string
  for opt := range optSet {
```

```
    optChoices = append(optChoices, opt)
   }
   sort.Strings(optChoices)
   optIndex := scenario.questioner.AskChoice(
    "The available micro DB instance classes for your database engine are:\n",
  optChoices)
   storageType := "standard"
   allocatedStorage := int32(5)
   log.Printf("Creating a DB instance named %v and database %v.\n"+
    "The DB instance is configured to use your custom parameter group %v,\n"+
    "selected engine %v,\n"+
    "selected DB instance class %v,"+
    "and %v GiB of %v storage.\n"+
    "This typically takes several minutes.",
    instanceName, dbName, *parameterGroup.DBParameterGroupName,
  *engineSelection.EngineVersion,
    optChoices[optIndex], allocatedStorage, storageType)
   instance, err = scenario.instances.CreateInstance(
    ctx, instanceName, dbName, *engineSelection.Engine,
  *engineSelection.EngineVersion,
    *parameterGroup.DBParameterGroupName, optChoices[optIndex], storageType,
    allocatedStorage, adminUsername, adminPassword)
   if err != nil {
    panic(err)
   }
   for *instance.DBInstanceStatus != "available" {
    scenario.helper.Pause(30)
    instance, err = scenario.instances.GetInstance(ctx, instanceName)
    if err != nil {
     panic(err)
    }
   }
   log.Println("Instance created and available.")
  }
  log.Println("Instance data:")
  log.Printf("\tDBInstanceIdentifier: %v\n", *instance.DBInstanceIdentifier)
  log.Printf("\tARN: %v\n", *instance.DBInstanceArn)
  log.Printf("\tStatus: %v\n", *instance.DBInstanceStatus)
  log.Printf("\tEngine: %v\n", *instance.Engine)
  log.Printf("\tEngine version: %v\n", *instance.EngineVersion)
  log.Println(strings.Repeat("-", 88))
  return instance
}
```

```
// DisplayConnection displays connection information about a DB instance and tips
// on how to connect to it.
func (scenario GetStartedInstances) DisplayConnection(instance *types.DBInstance) {
 log.Println(
  "You can now connect to your database by using your favorite MySQL client.\n" +
   "One way to connect is by using the 'mysql' shell on an Amazon EC2 instance\n" +
   "that is running in the same VPC as your DB instance. Pass the endpoint,\n" +
   "port, and administrator username to 'mysql'. Then, enter your password\n" +
   "when prompted:")
 log.Printf("\n\tmysql -h %v -P %v -u %v -p\n",
  *instance.Endpoint.Address, instance.Endpoint.Port, *instance.MasterUsername)
 log.Println("For more information, see the User Guide for RDS:\n" +
  "\thttps://docs.aws.amazon.com/AmazonRDS/latest/UserGuide/
CHAP_GettingStarted.CreatingConnecting.MySQL.html#CHAP_GettingStarted.Connecting.MySQL")
 log.Println(strings.Repeat("-", 88))
}

// CreateSnapshot shows how to create a DB instance snapshot and wait until it's
 available.
func (scenario GetStartedInstances) CreateSnapshot(ctx context.Context, instance
 *types.DBInstance) {
 if scenario.questioner.AskBool(
  "Do you want to create a snapshot of your DB instance (y/n)? ", "y") {
  snapshotId := fmt.Sprintf("%v-%v", *instance.DBInstanceIdentifier,
 scenario.helper.UniqueId())
  log.Printf("Creating a snapshot named %v. This typically takes a few minutes.\n",
 snapshotId)
  snapshot, err := scenario.instances.CreateSnapshot(ctx,
 *instance.DBInstanceIdentifier, snapshotId)
  if err != nil {
   panic(err)
  }
  for *snapshot.Status != "available" {
   scenario.helper.Pause(30)
   snapshot, err = scenario.instances.GetSnapshot(ctx, snapshotId)
   if err != nil {
    panic(err)
   }
  }
  log.Println("Snapshot data:")
  log.Printf("\tDBSnapshotIdentifier: %v\n", *snapshot.DBSnapshotIdentifier)
  log.Printf("\tARN: %v\n", *snapshot.DBSnapshotArn)
  log.Printf("\tStatus: %v\n", *snapshot.Status)
  log.Printf("\tEngine: %v\n", *snapshot.Engine)
```

```
  log.Printf("\tEngine version: %v\n", *snapshot.EngineVersion)
  log.Printf("\tDBInstanceIdentifier: %v\n", *snapshot.DBInstanceIdentifier)
  log.Printf("\tSnapshotCreateTime: %v\n", *snapshot.SnapshotCreateTime)
  log.Println(strings.Repeat("-", 88))
 }
}

// Cleanup shows how to clean up a DB instance and DB parameter group.
// Before the DB parameter group can be deleted, all associated DB instances must
 first be deleted.
func (scenario GetStartedInstances) Cleanup(
 ctx context.Context, instance *types.DBInstance, parameterGroup
 *types.DBParameterGroup) {

 if scenario.questioner.AskBool(
  "\nDo you want to delete the database instance and parameter group (y/n)? ", "y")
 {
  log.Printf("Deleting database instance %v.\n", *instance.DBInstanceIdentifier)
  err := scenario.instances.DeleteInstance(ctx, *instance.DBInstanceIdentifier)
  if err != nil {
   panic(err)
  }
  log.Println(
   "Waiting for the DB instance to delete. This typically takes several minutes.")
  for instance != nil {
   scenario.helper.Pause(30)
   instance, err = scenario.instances.GetInstance(ctx,
 *instance.DBInstanceIdentifier)
   if err != nil {
    panic(err)
   }
  }
  log.Printf("Deleting parameter group %v.", *parameterGroup.DBParameterGroupName)
  err = scenario.instances.DeleteParameterGroup(ctx,
 *parameterGroup.DBParameterGroupName)
  if err != nil {
   panic(err)
  }
 }
}

// IScenarioHelper abstracts the function from a scenario so that it
// can be mocked for unit testing.
type IScenarioHelper interface {
```

```
 Pause(secs int)
 UniqueId() string
}
type ScenarioHelper struct{}

// Pause waits for the specified number of seconds.
func (helper ScenarioHelper) Pause(secs int) {
 time.Sleep(time.Duration(secs) * time.Second)
}

// UniqueId returns a new UUID.
func (helper ScenarioHelper) UniqueId() string {
 return uuid.New().String()
}
```

Define functions that are called by the scenario to manage Amazon RDS actions.

```
import (
 "context"
 "errors"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}


// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
```

```
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBParameterGroups[0], err
 }
}



// CreateParameterGroup creates a DB parameter group that is based on the specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
  return output.DBParameterGroup, err
 }
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
```

```
    DBParameterGroupName: aws.String(parameterGroupName),
   })
  if err != nil {
   log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
   return err
  } else {
   return nil
  }
}



// GetParameters gets the parameters that are contained in a DB parameter group.
func (instances *DbInstances) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
```

```
    DBParameterGroupName: aws.String(parameterGroupName),
    Parameters:           params,
   })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}



// GetSnapshot gets a DB instance snapshot.
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName string)
 (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
```

```
 }
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string, dbInstanceClass
 string,
 storageType string, allocatedStorage int32, adminName string, adminPassword string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}



// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
```

```
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
```

```
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {

 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
   instanceOptions = append(instanceOptions, output.OrderableDBInstanceOptions...)
  }
 }
 return instanceOptions, err
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateDBInstance

- CreateDBParameterGroup
- CreateDBSnapshot
- DeleteDBInstance
- DeleteDBParameterGroup
- DescribeDBEngineVersions
- DescribeDBInstances
- DescribeDBParameterGroups
- DescribeDBParameters
- DescribeDBSnapshots
- DescribeOrderableDBInstanceOptions
- ModifyDBParameterGroup

Actions

`CreateDBInstance`

The following code example shows how to use `CreateDBInstance`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)
```

```
type DbInstances struct {
 RdsClient *rds.Client
}



// CreateInstance creates a DB instance.
func (instances *DbInstances) CreateInstance(ctx context.Context, instanceName
 string, dbName string,
 dbEngine string, dbEngineVersion string, parameterGroupName string, dbInstanceClass
 string,
 storageType string, allocatedStorage int32, adminName string, adminPassword string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.CreateDBInstance(ctx,
 &rds.CreateDBInstanceInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBName:               aws.String(dbName),
  DBParameterGroupName: aws.String(parameterGroupName),
  Engine:               aws.String(dbEngine),
  EngineVersion:        aws.String(dbEngineVersion),
  DBInstanceClass:      aws.String(dbInstanceClass),
  StorageType:          aws.String(storageType),
  AllocatedStorage:     aws.Int32(allocatedStorage),
  MasterUsername:       aws.String(adminName),
  MasterUserPassword:   aws.String(adminPassword),
 })
 if err != nil {
  log.Printf("Couldn't create instance %v: %v\n", instanceName, err)
  return nil, err
 } else {
  return output.DBInstance, nil
 }
}
```

- For API details, see CreateDBInstance in *AWS SDK for Go API Reference*.

`CreateDBParameterGroup`

The following code example shows how to use `CreateDBParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateParameterGroup creates a DB parameter group that is based on the specified
// parameter group family.
func (instances *DbInstances) CreateParameterGroup(
 ctx context.Context, parameterGroupName string, parameterGroupFamily string,
 description string) (
 *types.DBParameterGroup, error) {

 output, err := instances.RdsClient.CreateDBParameterGroup(ctx,
  &rds.CreateDBParameterGroupInput{
   DBParameterGroupName:   aws.String(parameterGroupName),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
   Description:            aws.String(description),
  })
 if err != nil {
  log.Printf("Couldn't create parameter group %v: %v\n", parameterGroupName, err)
  return nil, err
 } else {
```

```
  return output.DBParameterGroup, err
 }
}
```

- For API details, see CreateDBParameterGroup in *AWS SDK for Go API Reference*.

`CreateDBSnapshot`

The following code example shows how to use `CreateDBSnapshot`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// CreateSnapshot creates a snapshot of a DB instance.
func (instances *DbInstances) CreateSnapshot(ctx context.Context, instanceName
 string, snapshotName string) (
 *types.DBSnapshot, error) {
```

```
 output, err := instances.RdsClient.CreateDBSnapshot(ctx,
 &rds.CreateDBSnapshotInput{
  DBInstanceIdentifier: aws.String(instanceName),
  DBSnapshotIdentifier: aws.String(snapshotName),
 })
 if err != nil {
  log.Printf("Couldn't create snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return output.DBSnapshot, nil
 }
}
```

- For API details, see CreateDBSnapshot in *AWS SDK for Go API Reference*.

`DeleteDBInstance`

The following code example shows how to use `DeleteDBInstance`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
```

```
}



// DeleteInstance deletes a DB instance.
func (instances *DbInstances) DeleteInstance(ctx context.Context, instanceName
 string) error {
 _, err := instances.RdsClient.DeleteDBInstance(ctx, &rds.DeleteDBInstanceInput{
  DBInstanceIdentifier:   aws.String(instanceName),
  SkipFinalSnapshot:      aws.Bool(true),
  DeleteAutomatedBackups: aws.Bool(true),
 })
 if err != nil {
  log.Printf("Couldn't delete instance %v: %v\n", instanceName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBInstance in *AWS SDK for Go API Reference*.

`DeleteDBParameterGroup`

The following code example shows how to use `DeleteDBParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// DeleteParameterGroup deletes the named DB parameter group.
func (instances *DbInstances) DeleteParameterGroup(ctx context.Context,
 parameterGroupName string) error {
 _, err := instances.RdsClient.DeleteDBParameterGroup(ctx,
  &rds.DeleteDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  log.Printf("Couldn't delete parameter group %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see DeleteDBParameterGroup in *AWS SDK for Go API Reference*.

`DescribeDBEngineVersions`

The following code example shows how to use `DescribeDBEngineVersions`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetEngineVersions gets database engine versions that are available for the
 specified engine
// and parameter group family.
func (instances *DbInstances) GetEngineVersions(ctx context.Context, engine string,
 parameterGroupFamily string) (
 []types.DBEngineVersion, error) {
 output, err := instances.RdsClient.DescribeDBEngineVersions(ctx,
  &rds.DescribeDBEngineVersionsInput{
   Engine:                 aws.String(engine),
   DBParameterGroupFamily: aws.String(parameterGroupFamily),
  })
 if err != nil {
  log.Printf("Couldn't get engine versions for %v: %v\n", engine, err)
  return nil, err
 } else {
  return output.DBEngineVersions, nil
 }
}
```

- For API details, see DescribeDBEngineVersions in *AWS SDK for Go API Reference*.

`DescribeDBInstances`

The following code example shows how to use `DescribeDBInstances`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetInstance gets data about a DB instance.
func (instances *DbInstances) GetInstance(ctx context.Context, instanceName string)
 (
 *types.DBInstance, error) {
 output, err := instances.RdsClient.DescribeDBInstances(ctx,
  &rds.DescribeDBInstancesInput{
   DBInstanceIdentifier: aws.String(instanceName),
  })
 if err != nil {
  var notFoundError *types.DBInstanceNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("DB instance %v does not exist.\n", instanceName)
   err = nil
  } else {
```

```
   log.Printf("Couldn't get instance %v: %v\n", instanceName, err)
  }
  return nil, err
 } else {
  return &output.DBInstances[0], nil
 }
}
```

- For API details, see DescribeDBInstances in *AWS SDK for Go API Reference*.

`DescribeDBParameterGroups`

The following code example shows how to use `DescribeDBParameterGroups`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}
```

```
// GetParameterGroup gets a DB parameter group by name.
func (instances *DbInstances) GetParameterGroup(ctx context.Context,
 parameterGroupName string) (
 *types.DBParameterGroup, error) {
 output, err := instances.RdsClient.DescribeDBParameterGroups(
  ctx, &rds.DescribeDBParameterGroupsInput{
   DBParameterGroupName: aws.String(parameterGroupName),
  })
 if err != nil {
  var notFoundError *types.DBParameterGroupNotFoundFault
  if errors.As(err, &notFoundError) {
   log.Printf("Parameter group %v does not exist.\n", parameterGroupName)
   err = nil
  } else {
   log.Printf("Error getting parameter group %v: %v\n", parameterGroupName, err)
  }
  return nil, err
 } else {
  return &output.DBParameterGroups[0], err
 }
}
```

- For API details, see DescribeDBParameterGroups in *AWS SDK for Go API Reference*.

`DescribeDBParameters`

The following code example shows how to use `DescribeDBParameters`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetParameters gets the parameters that are contained in a DB parameter group.
func (instances *DbInstances) GetParameters(ctx context.Context, parameterGroupName
 string, source string) (
 []types.Parameter, error) {

 var output *rds.DescribeDBParametersOutput
 var params []types.Parameter
 var err error
 parameterPaginator := rds.NewDescribeDBParametersPaginator(instances.RdsClient,
  &rds.DescribeDBParametersInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Source:               aws.String(source),
  })
 for parameterPaginator.HasMorePages() {
  output, err = parameterPaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get parameters for %v: %v\n", parameterGroupName, err)
   break
  } else {
   params = append(params, output.Parameters...)
  }
 }
 return params, err
}
```

- For API details, see DescribeDBParameters in *AWS SDK for Go API Reference*.

`DescribeDBSnapshots`

The following code example shows how to use `DescribeDBSnapshots`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetSnapshot gets a DB instance snapshot.
func (instances *DbInstances) GetSnapshot(ctx context.Context, snapshotName string)
 (*types.DBSnapshot, error) {
 output, err := instances.RdsClient.DescribeDBSnapshots(ctx,
  &rds.DescribeDBSnapshotsInput{
   DBSnapshotIdentifier: aws.String(snapshotName),
  })
 if err != nil {
  log.Printf("Couldn't get snapshot %v: %v\n", snapshotName, err)
  return nil, err
 } else {
  return &output.DBSnapshots[0], nil
 }
}
```

- For API details, see DescribeDBSnapshots in *AWS SDK for Go API Reference*.

`DescribeOrderableDBInstanceOptions`

The following code example shows how to use `DescribeOrderableDBInstanceOptions`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// GetOrderableInstances uses a paginator to get DB instance options that can be
 used to create DB instances that are
// compatible with a set of specifications.
func (instances *DbInstances) GetOrderableInstances(ctx context.Context, engine
 string, engineVersion string) (
 []types.OrderableDBInstanceOption, error) {
```

```
 var output *rds.DescribeOrderableDBInstanceOptionsOutput
 var instanceOptions []types.OrderableDBInstanceOption
 var err error
 orderablePaginator :=
 rds.NewDescribeOrderableDBInstanceOptionsPaginator(instances.RdsClient,
  &rds.DescribeOrderableDBInstanceOptionsInput{
   Engine:        aws.String(engine),
   EngineVersion: aws.String(engineVersion),
  })
 for orderablePaginator.HasMorePages() {
  output, err = orderablePaginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get orderable DB instance options: %v\n", err)
   break
  } else {
   instanceOptions = append(instanceOptions, output.OrderableDBInstanceOptions...)
  }
 }
 return instanceOptions, err
}
```

- For API details, see DescribeOrderableDBInstanceOptions in *AWS SDK for Go API Reference*.

`ModifyDBParameterGroup`

The following code example shows how to use `ModifyDBParameterGroup`.

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/rds"
 "github.com/aws/aws-sdk-go-v2/service/rds/types"
)

type DbInstances struct {
 RdsClient *rds.Client
}



// UpdateParameters updates parameters in a named DB parameter group.
func (instances *DbInstances) UpdateParameters(ctx context.Context,
 parameterGroupName string, params []types.Parameter) error {
 _, err := instances.RdsClient.ModifyDBParameterGroup(ctx,
  &rds.ModifyDBParameterGroupInput{
   DBParameterGroupName: aws.String(parameterGroupName),
   Parameters:           params,
  })
 if err != nil {
  log.Printf("Couldn't update parameters in %v: %v\n", parameterGroupName, err)
  return err
 } else {
  return nil
 }
}
```

- For API details, see [ModifyDBParameterGroup](#) in *AWS SDK for Go API Reference*.

Serverless examples

Connecting to an Amazon RDS database in a Lambda function

The following code example shows how to implement a Lambda function that connects to an RDS database. The function makes a simple database request and returns the result.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Connecting to an Amazon RDS database in a Lambda function using Go.

```
/*
Golang v2 code here.
*/

package main

import (
 "context"
 "database/sql"
 "encoding/json"
 "fmt"
 "os"

 "github.com/aws/aws-lambda-go/lambda"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/feature/rds/auth"
 _ "github.com/go-sql-driver/mysql"
)

type MyEvent struct {
 Name string `json:"name"`
}

func HandleRequest(event *MyEvent) (map[string]interface{}, error) {

 var dbName string = os.Getenv("DatabaseName")
 var dbUser string = os.Getenv("DatabaseUser")
 var dbHost string = os.Getenv("DBHost") // Add hostname without https
 var dbPort int = os.Getenv("Port")      // Add port number
 var dbEndpoint string = fmt.Sprintf("%s:%d", dbHost, dbPort)
 var region string = os.Getenv("AWS_REGION")

 cfg, err := config.LoadDefaultConfig(context.TODO())
```

```
 if err != nil {
  panic("configuration error: " + err.Error())
 }

 authenticationToken, err := auth.BuildAuthToken(
  context.TODO(), dbEndpoint, region, dbUser, cfg.Credentials)
 if err != nil {
  panic("failed to create authentication token: " + err.Error())
 }

 dsn := fmt.Sprintf("%s:%s@tcp(%s)/%s?tls=true&allowCleartextPasswords=true",
  dbUser, authenticationToken, dbEndpoint, dbName,
 )

 db, err := sql.Open("mysql", dsn)
 if err != nil {
  panic(err)
 }

 defer db.Close()

 var sum int
 err = db.QueryRow("SELECT ?+? AS sum", 3, 2).Scan(&sum)
 if err != nil {
  panic(err)
 }
 s := fmt.Sprint(sum)
 message := fmt.Sprintf("The selected sum is: %s", s)

 messageBytes, err := json.Marshal(message)
 if err != nil {
  return nil, err
 }

 messageString := string(messageBytes)
 return map[string]interface{}{
  "statusCode": 200,
  "headers":    map[string]string{"Content-Type": "application/json"},
  "body":       messageString,
 }, nil
}

func main() {
 lambda.Start(HandleRequest)
```

```
}
```

Amazon Redshift examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon Redshift.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions

Get started

Hello Amazon Redshift

The following code example shows how to get started using Amazon Redshift.

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"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
)

// main uses the AWS SDK for Go V2 to create a Redshift client
// and list up to 10 clusters 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
 }
 redshiftClient := redshift.NewFromConfig(sdkConfig)
 count := 20
 fmt.Printf("Let's list up to %v clusters for your account.\n", count)
 result, err := redshiftClient.DescribeClusters(ctx,
 &redshift.DescribeClustersInput{
  MaxRecords: aws.Int32(int32(count)),
 })
 if err != nil {
  fmt.Printf("Couldn't list clusters for your account. Here's why: %v\n", err)
  return
 }
 if len(result.Clusters) == 0 {
  fmt.Println("You don't have any clusters!")
  return
 }
 for _, cluster := range result.Clusters {
  fmt.Printf("\t%v : %v\n", *cluster.ClusterIdentifier, *cluster.ClusterStatus)
 }
}
```

- For API details, see DescribeClusters in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to:

- Create a Redshift cluster.
- List databases in the cluster.
- Create a table named Movies.
- Populate the Movies table.
- Query the Movies table by year.
- Modify the Redshift cluster.
- Delete the Amazon Redshift cluster.

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 scenarios

import (
 "context"
 "encoding/json"
 "errors"
 "fmt"
 "log"
 "math/rand"
 "strings"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 redshift_types "github.com/aws/aws-sdk-go-v2/service/redshift/types"
 redshiftdata_types "github.com/aws/aws-sdk-go-v2/service/redshiftdata/types"
 "github.com/aws/aws-sdk-go-v2/service/secretsmanager"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/redshift/actions"

 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshiftdata"
)

// IScenarioHelper abstracts input and wait functions from a scenario so that they
// can be mocked for unit testing.
type IScenarioHelper interface {
 GetName() string
}

const rMax = 100000

type ScenarioHelper struct {
 Prefix string
 Random *rand.Rand
}

// GetName returns a unique name formed of a prefix and a random number.
func (helper ScenarioHelper) GetName() string {
 return fmt.Sprintf("%v%v", helper.Prefix, helper.Random.Intn(rMax))
}

// RedshiftBasicsScenario separates the steps of this scenario into individual
 functions so that
// they are simpler to read and understand.
type RedshiftBasicsScenario struct {
 sdkConfig         aws.Config
 helper            IScenarioHelper
 questioner        demotools.IQuestioner
 pauser            demotools.IPausable
 filesystem        demotools.IFileSystem
 redshiftActor     *actions.RedshiftActions
 redshiftDataActor *actions.RedshiftDataActions
 secretsmanager    *SecretsManager
}

// SecretsManager is used to retrieve username and password information from a
 secure service.
type SecretsManager struct {
 SecretsManagerClient *secretsmanager.Client
}
```

```
// RedshiftBasics constructs a new Redshift Basics runner.
func RedshiftBasics(sdkConfig aws.Config, questioner demotools.IQuestioner, pauser
 demotools.IPausable, filesystem demotools.IFileSystem, helper IScenarioHelper)
 RedshiftBasicsScenario {
 scenario := RedshiftBasicsScenario{
  sdkConfig:         sdkConfig,
  helper:            helper,
  questioner:        questioner,
  pauser:            pauser,
  filesystem:        filesystem,
  secretsmanager:    &SecretsManager{SecretsManagerClient:
 secretsmanager.NewFromConfig(sdkConfig)},
  redshiftActor:     &actions.RedshiftActions{RedshiftClient:
 redshift.NewFromConfig(sdkConfig)},
  redshiftDataActor: &actions.RedshiftDataActions{RedshiftDataClient:
 redshiftdata.NewFromConfig(sdkConfig)},
 }
 return scenario
}


// Movie makes it easier to use Movie objects given in json format.
type Movie struct {
 ID    int    `json:"id"`
 Title string `json:"title"`
 Year  int    `json:"year"`
}


// User makes it easier to get the User data back from SecretsManager and use it
 later.
type User struct {
 Username string `json:"userName"`
 Password string `json:"userPassword"`
}

// Run runs the RedshiftBasics interactive example that shows you how to use Amazon
// Redshift and how to interact with its common endpoints.
//
// 0. Retrieve username and password information to access Redshift.
// 1. Create a cluster.
// 2. Wait for the cluster to become available.
// 3. List the available databases in the region.
```

```
// 4. Create a table named "Movies" in the "dev" database.
// 5. Populate the movies table from the "movies.json" file.
// 6. Query the movies table by year.
// 7. Modify the cluster's maintenance window.
// 8. Optionally clean up all resources created during this demo.
//
// This example creates an Amazon Redshift service client from the specified
 sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func (runner *RedshiftBasicsScenario) Run(ctx context.Context) {

 user := User{}
 secretId := "s3express/basics/secrets"
 clusterId := "demo-cluster-1"
 maintenanceWindow := "wed:07:30-wed:08:00"
 databaseName := "dev"
 tableName := "Movies"
 fileName := "Movies.json"
 nodeType := "ra3.xlplus"
 clusterType := "single-node"

 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := runner.questioner.(*demotools.MockQuestioner)
   if isMock || runner.questioner.AskBool("Do you want to see the full error message
(y/n)?", "y") {
    log.Println(r)
   }
   runner.cleanUpResources(ctx, clusterId, databaseName, tableName, user.Username,
runner.questioner)
  }
 }()

 // Retrieve the userName and userPassword from SecretsManager
 output, err := runner.secretsmanager.SecretsManagerClient.GetSecretValue(ctx,
 &secretsmanager.GetSecretValueInput{
  SecretId: aws.String(secretId),
 })
 if err != nil {
```

```
 log.Printf("There was a problem getting the secret value: %s", err)
 log.Printf("Please make sure to create a secret named 's3express/basics/secrets'
with keys of 'userName' and 'userPassword'.")
 panic(err)
}

err = json.Unmarshal([]byte(*output.SecretString), &user)
if err != nil {
 log.Printf("There was a problem parsing the secret value from JSON: %s", err)
 panic(err)
}

// Create the Redshift cluster
_, err = runner.redshiftActor.CreateCluster(ctx, clusterId, user.Username,
user.Password, nodeType, clusterType, true)
if err != nil {
 var clusterAlreadyExistsFault *redshift_types.ClusterAlreadyExistsFault
 if errors.As(err, &clusterAlreadyExistsFault) {
  log.Println("Cluster already exists. Continuing.")
 } else {
  log.Println("Error creating cluster.")
  panic(err)
 }
}

// Wait for the cluster to become available
waiter := redshift.NewClusterAvailableWaiter(runner.redshiftActor.RedshiftClient)
err = waiter.Wait(ctx, &redshift.DescribeClustersInput{
 ClusterIdentifier: aws.String(clusterId),
}, 5*time.Minute)
if err != nil {
 log.Println("An error occurred waiting for the cluster.")
 panic(err)
}

// Get some info about the cluster
describeOutput, err := runner.redshiftActor.DescribeClusters(ctx, clusterId)
if err != nil {
 log.Println("Something went wrong trying to get information about the cluster.")
 panic(err)
}
log.Println("Here's some information about the cluster.")
log.Printf("The cluster's status is %s", *describeOutput.Clusters[0].ClusterStatus)
```

```
log.Printf("The cluster was created at %s",
*describeOutput.Clusters[0].ClusterCreateTime)

// List databases
log.Println("List databases in", clusterId)
runner.questioner.Ask("Press Enter to continue...")
err = runner.redshiftDataActor.ListDatabases(ctx, clusterId, databaseName,
user.Username)
if err != nil {
 log.Printf("Failed to list databases: %v\n", err)
 panic(err)
}

// Create the "Movies" table
log.Println("Now you will create a table named " + tableName + ".")
runner.questioner.Ask("Press Enter to continue...")
err = nil
result, err := runner.redshiftDataActor.CreateTable(ctx, clusterId, databaseName,
tableName, user.Username, runner.pauser, []string{"title VARCHAR(256)", "year
INT"})
if err != nil {
 log.Printf("Failed to create table: %v\n", err)
 panic(err)
}

describeInput := redshiftdata.DescribeStatementInput{
 Id: result.Id,
}
query := actions.RedshiftQuery{
 Context: ctx,
 Input:   describeInput,
 Result:  result,
}
err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
if err != nil {
 log.Printf("Failed to execute query: %v\n", err)
 panic(err)
}
log.Printf("Successfully executed query\n")

// Populate the "Movies" table
runner.PopulateMoviesTable(ctx, clusterId, databaseName, tableName, user.Username,
fileName)
```

```
 // Query the "Movies" table by year
 log.Println("Query the Movies table by year.")
 year := runner.questioner.AskInt(
  fmt.Sprintf("Enter a value between %v and %v:", 2012, 2014),
  demotools.InIntRange{Lower: 2012, Upper: 2014})
 runner.QueryMoviesByYear(ctx, clusterId, databaseName, tableName, user.Username,
 year)

 // Modify the cluster's maintenance window
 runner.redshiftActor.ModifyCluster(ctx, clusterId, maintenanceWindow)

 // Delete the Redshift cluster if confirmed
 runner.cleanUpResources(ctx, clusterId, databaseName, tableName, user.Username,
 runner.questioner)

 log.Println("Thanks for watching!")
}

// cleanUpResources asks the user if they would like to delete each resource created
 during the scenario, from most
// impactful to least impactful. If any choice to delete is made, further deletion
 attempts are skipped.
func (runner *RedshiftBasicsScenario) cleanUpResources(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string,
 questioner demotools.IQuestioner) {
 deleted := false
 var err error = nil
 if questioner.AskBool("Do you want to delete the entire cluster? This will clean up
 all resources. (y/n)", "y") {
  deleted, err = runner.redshiftActor.DeleteCluster(ctx, clusterId)
  if err != nil {
   log.Printf("Error deleting cluster: %v", err)
  }
 }
 if !deleted && questioner.AskBool("Do you want to delete the dev table? This will
 clean up all inserted records but keep your cluster intact. (y/n)", "y") {
  deleted, err = runner.redshiftDataActor.DeleteTable(ctx, clusterId, databaseName,
 tableName, userName)
  if err != nil {
   log.Printf("Error deleting movies table: %v", err)
  }
 }
```

```
 if !deleted && questioner.AskBool("Do you want to delete all rows in the Movies
 table? This will clean up all inserted records but keep your cluster and table
 intact. (y/n)", "y") {
  deleted, err = runner.redshiftDataActor.DeleteDataRows(ctx, clusterId,
 databaseName, tableName, userName, runner.pauser)
  if err != nil {
   log.Printf("Error deleting data rows: %v", err)
  }
 }
 if !deleted {
  log.Print("Please manually delete any unwanted resources.")
 }
}


// loadMoviesFromJSON takes the <fileName> file and populates a slice of Movie
 objects.
func (runner *RedshiftBasicsScenario) loadMoviesFromJSON(fileName string, filesystem
 demotools.IFileSystem) ([]Movie, error) {
 file, err := filesystem.OpenFile("../../resources/sample_files/" + fileName)
 if err != nil {
  return nil, err
 }
 defer filesystem.CloseFile(file)

 var movies []Movie
 err = json.NewDecoder(file).Decode(&movies)
 if err != nil {
  return nil, err
 }

 return movies, nil
}



// PopulateMoviesTable reads data from the <fileName> file and inserts records into
 the "Movies" table.
func (runner *RedshiftBasicsScenario) PopulateMoviesTable(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string, fileName
 string) {
 log.Println("Populate the " + tableName + " table using the " + fileName + "
 file.")
 numRecords := runner.questioner.AskInt(
```

```
 fmt.Sprintf("Enter a value between %v and %v:", 10, 100),
 demotools.InIntRange{Lower: 10, Upper: 100})

movies, err := runner.loadMoviesFromJSON(fileName, runner.filesystem)
if err != nil {
 log.Printf("Failed to load movies from JSON: %v\n", err)
 panic(err)
}

var sqlStatements []string

for i, movie := range movies {
 if i >= numRecords {
  break
 }

 sqlStatement := fmt.Sprintf(`INSERT INTO %s (title, year) VALUES ('%s', %d);`,
  tableName,
  strings.Replace(movie.Title, "'", "''", -1), // Double any single quotes to
escape them
  movie.Year)

 sqlStatements = append(sqlStatements, sqlStatement)
}

input := &redshiftdata.BatchExecuteStatementInput{
 ClusterIdentifier: aws.String(clusterId),
 Database:          aws.String(databaseName),
 DbUser:            aws.String(userName),
 Sqls:              sqlStatements,
}

result, err := runner.redshiftDataActor.ExecuteBatchStatement(ctx, *input)
if err != nil {
 log.Printf("Failed to execute batch statement: %v\n", err)
 panic(err)
}

describeInput := redshiftdata.DescribeStatementInput{
 Id: result.Id,
}

query := actions.RedshiftQuery{
 Context: ctx,
```

```
  Result:  result,
  Input:   describeInput,
 }
 err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
 if err != nil {
  log.Printf("Failed to execute batch insert query: %v\n", err)
  return
 }
 log.Printf("Successfully executed batch statement\n")

 log.Printf("%d records were added to the Movies table.\n", numRecords)
}



// QueryMoviesByYear retrieves only movies from the "Movies" table which match the
 given year.
func (runner *RedshiftBasicsScenario) QueryMoviesByYear(ctx context.Context,
 clusterId string, databaseName string, tableName string, userName string, year int)
 {

 sqlStatement := fmt.Sprintf(`SELECT title FROM %s WHERE year = %d;`, tableName,
 year)

 input := &redshiftdata.ExecuteStatementInput{
  ClusterIdentifier: aws.String(clusterId),
  Database:          aws.String(databaseName),
  DbUser:            aws.String(userName),
  Sql:               aws.String(sqlStatement),
 }

 result, err := runner.redshiftDataActor.ExecuteStatement(ctx, *input)
 if err != nil {
  log.Printf("Failed to query movies: %v\n", err)
  panic(err)
 }

 log.Println("The identifier of the statement is ", *result.Id)

 describeInput := redshiftdata.DescribeStatementInput{
  Id: result.Id,
 }

 query := actions.RedshiftQuery{
```

```
  Context: ctx,
  Input:   describeInput,
  Result:  result,
 }
 err = runner.redshiftDataActor.WaitForQueryStatus(query, runner.pauser, true)
 if err != nil {
  log.Printf("Failed to execute query: %v\n", err)
  panic(err)
 }
 log.Printf("Successfully executed query\n")

 getResultOutput, err := runner.redshiftDataActor.GetStatementResult(ctx,
 *result.Id)
 if err != nil {
  log.Printf("Failed to query movies: %v\n", err)
  panic(err)
 }
 for _, row := range getResultOutput.Records {
  for _, col := range row {
   title, ok := col.(*redshiftdata_types.FieldMemberStringValue)
   if !ok {
    log.Println("Failed to parse the field")
   } else {
    log.Printf("The Movie title field is %s\n", title.Value)
   }
  }
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateCluster
  - DescribeClusters
  - DescribeStatement
  - ExecuteStatement
  - GetStatementResult
  - ListDatabasesPaginator
  - ModifyCluster

Actions

`CreateCluster`

The following code example shows how to use `CreateCluster`.

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"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// CreateCluster sends a request to create a cluster with the given clusterId using
 the provided credentials.
func (actor RedshiftActions) CreateCluster(ctx context.Context, clusterId string,
 userName string, userPassword string, nodeType string, clusterType string,
 publiclyAccessible bool) (*redshift.CreateClusterOutput, error) {
 // Create a new Redshift cluster
 input := &redshift.CreateClusterInput{
```

```
  ClusterIdentifier:  aws.String(clusterId),
  MasterUserPassword: aws.String(userPassword),
  MasterUsername:     aws.String(userName),
  NodeType:           aws.String(nodeType),
  ClusterType:        aws.String(clusterType),
  PubliclyAccessible: aws.Bool(publiclyAccessible),
 }
 var opErr *types.ClusterAlreadyExistsFault
 output, err := actor.RedshiftClient.CreateCluster(ctx, input)
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster already exists")
  return nil, nil
 } else if err != nil {
  log.Printf("Failed to create Redshift cluster: %v\n", err)
  return nil, err
 }

 log.Printf("Created cluster %s\n", *output.Cluster.ClusterIdentifier)
 return output, nil
}
```

- For API details, see CreateCluster in *AWS SDK for Go API Reference*.

`DeleteCluster`

The following code example shows how to use `DeleteCluster`.

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"
```

```
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// DeleteCluster deletes the given cluster.
func (actor RedshiftActions) DeleteCluster(ctx context.Context, clusterId string)
 (bool, error) {
 input := redshift.DeleteClusterInput{
  ClusterIdentifier:        aws.String(clusterId),
  SkipFinalClusterSnapshot: aws.Bool(true),
 }
 _, err := actor.RedshiftClient.DeleteCluster(ctx, &input)
 var opErr *types.ClusterNotFoundFault
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster was not found. Where could it be?")
  return false, err
 } else if err != nil {
  log.Printf("Failed to delete Redshift cluster: %v\n", err)
  return false, err
 }
 waiter := redshift.NewClusterDeletedWaiter(actor.RedshiftClient)
 err = waiter.Wait(ctx, &redshift.DescribeClustersInput{
  ClusterIdentifier: aws.String(clusterId),
 }, 5*time.Minute)
 if err != nil {
  log.Printf("Wait time exceeded for deleting cluster, continuing: %v\n", err)
 }
 log.Printf("The cluster %s was deleted\n", clusterId)
 return true, nil
}
```

- For API details, see DeleteCluster in *AWS SDK for Go API Reference*.

`DescribeClusters`

The following code example shows how to use `DescribeClusters`.

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"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)



// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// DescribeClusters returns information about the given cluster.
func (actor RedshiftActions) DescribeClusters(ctx context.Context, clusterId string)
 (*redshift.DescribeClustersOutput, error) {
 input, err := actor.RedshiftClient.DescribeClusters(ctx,
 &redshift.DescribeClustersInput{
```

```
  ClusterIdentifier: aws.String(clusterId),
 })
 var opErr *types.AccessToClusterDeniedFault
 if errors.As(err, &opErr) {
  println("Access to cluster denied.")
  panic(err)
 } else if err != nil {
  println("Failed to describe Redshift clusters.")
  return nil, err
 }
 return input, nil
}
```

- For API details, see DescribeClusters in *AWS SDK for Go API Reference*.

`ModifyCluster`

The following code example shows how to use `ModifyCluster`.

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"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/redshift"
 "github.com/aws/aws-sdk-go-v2/service/redshift/types"
)
```

```
// RedshiftActions wraps Redshift service actions.
type RedshiftActions struct {
 RedshiftClient *redshift.Client
}



// ModifyCluster sets the preferred maintenance window for the given cluster.
func (actor RedshiftActions) ModifyCluster(ctx context.Context, clusterId string,
 maintenanceWindow string) *redshift.ModifyClusterOutput {
 // Modify the cluster's maintenance window
 input := &redshift.ModifyClusterInput{
  ClusterIdentifier:          aws.String(clusterId),
  PreferredMaintenanceWindow: aws.String(maintenanceWindow),
 }

 var opErr *types.InvalidClusterStateFault
 output, err := actor.RedshiftClient.ModifyCluster(ctx, input)
 if err != nil && errors.As(err, &opErr) {
  log.Println("Cluster is in an invalid state.")
  panic(err)
 } else if err != nil {
  log.Printf("Failed to modify Redshift cluster: %v\n", err)
  panic(err)
 }

 log.Printf("The cluster was successfully modified and now has %s as the maintenance
 window\n", *output.Cluster.PreferredMaintenanceWindow)
 return output
}
```

- For API details, see ModifyCluster in *AWS SDK for Go API Reference*.

Amazon S3 examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon S3.

Basics are code examples that show you how to perform the essential operations within a service.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Basics
- Actions
- Scenarios
- Serverless examples

Get started

Hello Amazon S3

The following code example shows how to get started using Amazon S3.

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"
 "errors"
 "fmt"

 "github.com/aws/aws-sdk-go-v2/config"
```

```
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/smithy-go"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Storage Service
// (Amazon S3) client and list up to 10 buckets 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
 }
 s3Client := s3.NewFromConfig(sdkConfig)
 count := 10
 fmt.Printf("Let's list up to %v buckets for your account.\n", count)
 result, err := s3Client.ListBuckets(ctx, &s3.ListBucketsInput{})
 if err != nil {
  var ae smithy.APIError
  if errors.As(err, &ae) && ae.ErrorCode() == "AccessDenied" {
   fmt.Println("You don't have permission to list buckets for this account.")
  } else {
   fmt.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
  }
  return
 }
 if len(result.Buckets) == 0 {
  fmt.Println("You don't have any buckets!")
 } else {
  if count > len(result.Buckets) {
   count = len(result.Buckets)
  }
  for _, bucket := range result.Buckets[:count] {
   fmt.Printf("\t%v\n", *bucket.Name)
  }
 }
}
```

- For API details, see ListBuckets in *AWS SDK for Go API Reference*.

Basics

Learn the basics

The following code example shows how to:

- Create a bucket and upload a file to it.
- Download an object from a bucket.
- Copy an object to a subfolder in a bucket.
- List the objects in a bucket.
- Delete the bucket objects and the bucket.

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.

Define a struct that wraps bucket and object actions used by the scenario.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
```

```
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListBuckets lists the buckets in the current account.
func (basics BucketBasics) ListBuckets(ctx context.Context) ([]types.Bucket, error)
 {
 var err error
 var output *s3.ListBucketsOutput
 var buckets []types.Bucket
 bucketPaginator := s3.NewListBucketsPaginator(basics.S3Client,
 &s3.ListBucketsInput{})
 for bucketPaginator.HasMorePages() {
  output, err = bucketPaginator.NextPage(ctx)
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "AccessDenied" {
    fmt.Println("You don't have permission to list buckets for this account.")
    err = apiErr
   } else {
    log.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
   }
   break
  } else {
   buckets = append(buckets, output.Buckets...)
  }
 }
 return buckets, err
}



// BucketExists checks whether a bucket exists in the current account.
func (basics BucketBasics) BucketExists(ctx context.Context, bucketName string)
 (bool, error) {
```

```
 _, err := basics.S3Client.HeadBucket(ctx, &s3.HeadBucketInput{
  Bucket: aws.String(bucketName),
 })
 exists := true
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NotFound:
    log.Printf("Bucket %v is available.\n", bucketName)
    exists = false
    err = nil
   default:
    log.Printf("Either you don't have access to bucket %v or another error occurred.
 "+
     "Here's what happened: %v\n", bucketName, err)
   }
  }
 } else {
  log.Printf("Bucket %v exists and you already own it.", bucketName)
 }

 return exists, err
}



// CreateBucket creates a bucket with the specified name in the specified Region.
func (basics BucketBasics) CreateBucket(ctx context.Context, name string, region
 string) error {
 _, err := basics.S3Client.CreateBucket(ctx, &s3.CreateBucketInput{
  Bucket: aws.String(name),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 })
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", name)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", name)
```

```
    err = exists
   }
  } else {
   err = s3.NewBucketExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadBucketInput{Bucket: aws.String(name)}, time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for bucket %s to exist.\n", name)
   }
  }
  return err
}



// UploadFile reads from a file and puts the data into an object in a bucket.
func (basics BucketBasics) UploadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 file, err := os.Open(fileName)
 if err != nil {
  log.Printf("Couldn't open file %v to upload. Here's why: %v\n", fileName, err)
 } else {
  defer file.Close()
  _, err = basics.S3Client.PutObject(ctx, &s3.PutObjectInput{
   Bucket: aws.String(bucketName),
   Key:    aws.String(objectKey),
   Body:   file,
  })
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
    log.Printf("Error while uploading object to %s. The object is too large.\n"+
     "To upload objects larger than 5GB, use the S3 console (160GB max)\n"+
     "or the multipart upload API (5TB max).", bucketName)
   } else {
    log.Printf("Couldn't upload file %v to %v:%v. Here's why: %v\n",
     fileName, bucketName, objectKey, err)
   }
  } else {
   err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
   }
```

```
  }
 }
 return err
}



// UploadLargeObject uses an upload manager to upload data to an object in a bucket.
// The upload manager breaks large data into parts and uploads the parts
 concurrently.
func (basics BucketBasics) UploadLargeObject(ctx context.Context, bucketName string,
 objectKey string, largeObject []byte) error {
 largeBuffer := bytes.NewReader(largeObject)
 var partMiBs int64 = 10
 uploader := manager.NewUploader(basics.S3Client, func(u *manager.Uploader) {
  u.PartSize = partMiBs * 1024 * 1024
 })
 _, err := uploader.Upload(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
  Body:   largeBuffer,
 })
 if err != nil {
  var apiErr smithy.APIError
  if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
   log.Printf("Error while uploading object to %s. The object is too large.\n"+
    "The maximum size for a multipart upload is 5TB.", bucketName)
  } else {
   log.Printf("Couldn't upload large object to %v:%v. Here's why: %v\n",
    bucketName, objectKey, err)
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }

 return err
}
```

```
// DownloadFile gets an object from a bucket and stores it in a local file.
func (basics BucketBasics) DownloadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 result, err := basics.S3Client.GetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Can't get object %s from bucket %s. No such key exists.\n",
 objectKey, bucketName)
   err = noKey
  } else {
   log.Printf("Couldn't get object %v:%v. Here's why: %v\n", bucketName, objectKey,
 err)
  }
  return err
 }
 defer result.Body.Close()
 file, err := os.Create(fileName)
 if err != nil {
  log.Printf("Couldn't create file %v. Here's why: %v\n", fileName, err)
  return err
 }
 defer file.Close()
 body, err := io.ReadAll(result.Body)
 if err != nil {
  log.Printf("Couldn't read object body from %v. Here's why: %v\n", objectKey, err)
 }
 _, err = file.Write(body)
 return err
}



// DownloadLargeObject uses a download manager to download an object from a bucket.
// The download manager gets the data in parts and writes them to a buffer until all
 of
// the data has been downloaded.
func (basics BucketBasics) DownloadLargeObject(ctx context.Context, bucketName
 string, objectKey string) ([]byte, error) {
 var partMiBs int64 = 10
```

```
 downloader := manager.NewDownloader(basics.S3Client, func(d *manager.Downloader) {
  d.PartSize = partMiBs * 1024 * 1024
 })
 buffer := manager.NewWriteAtBuffer([]byte{})
 _, err := downloader.Download(ctx, buffer, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  log.Printf("Couldn't download large object from %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return buffer.Bytes(), err
}



// CopyToFolder copies an object in a bucket to a subfolder in the same bucket.
func (basics BucketBasics) CopyToFolder(ctx context.Context, bucketName string,
 objectKey string, folderName string) error {
 objectDest := fmt.Sprintf("%v/%v", folderName, objectKey)
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(bucketName),
  CopySource: aws.String(fmt.Sprintf("%v/%v", bucketName, objectKey)),
  Key:        aws.String(objectDest),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, bucketName)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectDest)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectDest)
  }
 }
 return err
}
```

```
// CopyToBucket copies an object in a bucket to another bucket.
func (basics BucketBasics) CopyToBucket(ctx context.Context, sourceBucket string,
 destinationBucket string, objectKey string) error {
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(destinationBucket),
  CopySource: aws.String(fmt.Sprintf("%v/%v", sourceBucket, objectKey)),
  Key:        aws.String(objectKey),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, sourceBucket)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(destinationBucket), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }
 return err
}



// ListObjects lists the objects in a bucket.
func (basics BucketBasics) ListObjects(ctx context.Context, bucketName string)
 ([]types.Object, error) {
 var err error
 var output *s3.ListObjectsV2Output
 input := &s3.ListObjectsV2Input{
  Bucket: aws.String(bucketName),
 }
 var objects []types.Object
 objectPaginator := s3.NewListObjectsV2Paginator(basics.S3Client, input)
 for objectPaginator.HasMorePages() {
  output, err = objectPaginator.NextPage(ctx)
```

```
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
   break
  } else {
   objects = append(objects, output.Contents...)
  }
 }
 return objects, err
}



// DeleteObjects deletes a list of objects from a bucket.
func (basics BucketBasics) DeleteObjects(ctx context.Context, bucketName string,
 objectKeys []string) error {
 var objectIds []types.ObjectIdentifier
 for _, key := range objectKeys {
  objectIds = append(objectIds, types.ObjectIdentifier{Key: aws.String(key)})
 }
 output, err := basics.S3Client.DeleteObjects(ctx, &s3.DeleteObjectsInput{
  Bucket: aws.String(bucketName),
  Delete: &types.Delete{Objects: objectIds, Quiet: aws.Bool(true)},
 })
 if err != nil || len(output.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucketName)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
  } else if len(output.Errors) > 0 {
   for _, outErr := range output.Errors {
    log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
   }
   err = fmt.Errorf("%s", *output.Errors[0].Message)
  }
 } else {
  for _, delObjs := range output.Deleted {
   err = s3.NewObjectNotExistsWaiter(basics.S3Client).Wait(
```

```
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key: delObjs.Key},
 time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to be deleted.\n",
 *delObjs.Key)
   } else {
    log.Printf("Deleted %s.\n", *delObjs.Key)
   }
  }
 }
 return err
}



// DeleteBucket deletes a bucket. The bucket must be empty or an error is returned.
func (basics BucketBasics) DeleteBucket(ctx context.Context, bucketName string)
 error {
 _, err := basics.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
  Bucket: aws.String(bucketName)})
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucketName)
   err = noBucket
  } else {
   log.Printf("Couldn't delete bucket %v. Here's why: %v\n", bucketName, err)
  }
 } else {
  err = s3.NewBucketNotExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucketName)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucketName)
  } else {
   log.Printf("Deleted %s.\n", bucketName)
  }
 }
 return err
}
```

Run an interactive scenario that shows you how to work with S3 buckets and objects.

```
import (
 "context"
 "fmt"
 "log"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)

// RunGetStartedScenario is an interactive example that shows you how to use Amazon
// Simple Storage Service (Amazon S3) to create an S3 bucket and use it to store
 objects.
//
// 1. Create a bucket.
// 2. Upload a local file to the bucket.
// 3. Download an object to a local file.
// 4. Copy an object to a different folder in the bucket.
// 5. List objects in the bucket.
// 6. Delete all objects in the bucket.
// 7. Delete the bucket.
//
// This example creates an Amazon S3 service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunGetStartedScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
```

```
 }
}()

log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon S3 getting started demo.")
log.Println(strings.Repeat("-", 88))

s3Client := s3.NewFromConfig(sdkConfig)
bucketBasics := actions.BucketBasics{S3Client: s3Client}

count := 10
log.Printf("Let's list up to %v buckets for your account:", count)
buckets, err := bucketBasics.ListBuckets(ctx)
if err != nil {
 panic(err)
}
if len(buckets) == 0 {
 log.Println("You don't have any buckets!")
} else {
 if count > len(buckets) {
  count = len(buckets)
 }
 for _, bucket := range buckets[:count] {
  log.Printf("\t%v\n", *bucket.Name)
 }
}

bucketName := questioner.Ask("Let's create a bucket. Enter a name for your
bucket:",
 demotools.NotEmpty{})
bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
if err != nil {
 panic(err)
}
if !bucketExists {
 err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
 if err != nil {
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))
```

```
fmt.Println("Let's upload a file to your bucket.")
smallFile := questioner.Ask("Enter the path to a file you want to upload:",
 demotools.NotEmpty{})
const smallKey = "doc-example-key"
err = bucketBasics.UploadFile(ctx, bucketName, smallKey, smallFile)
if err != nil {
 panic(err)
}
log.Printf("Uploaded %v as %v.\n", smallFile, smallKey)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's download %v to a file.", smallKey)
downloadFileName := questioner.Ask("Enter a name for the downloaded file:",
demotools.NotEmpty{})
err = bucketBasics.DownloadFile(ctx, bucketName, smallKey, downloadFileName)
if err != nil {
 panic(err)
}
log.Printf("File %v downloaded.", downloadFileName)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's copy %v to a folder in the same bucket.", smallKey)
folderName := questioner.Ask("Enter a folder name: ", demotools.NotEmpty{})
err = bucketBasics.CopyToFolder(ctx, bucketName, smallKey, folderName)
if err != nil {
 panic(err)
}
log.Printf("Copied %v to %v/%v.\n", smallKey, folderName, smallKey)
log.Println(strings.Repeat("-", 88))

log.Println("Let's list the objects in your bucket.")
questioner.Ask("Press Enter when you're ready.")
objects, err := bucketBasics.ListObjects(ctx, bucketName)
if err != nil {
 panic(err)
}
log.Printf("Found %v objects.\n", len(objects))
var objKeys []string
for _, object := range objects {
 objKeys = append(objKeys, *object.Key)
 log.Printf("\t%v\n", *object.Key)
}
log.Println(strings.Repeat("-", 88))
```

```
 if questioner.AskBool("Do you want to delete your bucket and all of its "+
  "contents? (y/n)", "y") {
  log.Println("Deleting objects.")
  err = bucketBasics.DeleteObjects(ctx, bucketName, objKeys)
  if err != nil {
   panic(err)
  }
  log.Println("Deleting bucket.")
  err = bucketBasics.DeleteBucket(ctx, bucketName)
  if err != nil {
   panic(err)
  }
  log.Printf("Deleting downloaded file %v.\n", downloadFileName)
  err = os.Remove(downloadFileName)
  if err != nil {
   panic(err)
  }
 } else {
  log.Println("Okay. Don't forget to delete objects from your bucket to avoid
 charges.")
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CopyObject
  - CreateBucket
  - DeleteBucket
  - DeleteObjects
  - GetObject
  - ListObjectsV2
  - PutObject

Actions

`CopyObject`

The following code example shows how to use `CopyObject`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}
```

```
// CopyToBucket copies an object in a bucket to another bucket.
func (basics BucketBasics) CopyToBucket(ctx context.Context, sourceBucket string,
 destinationBucket string, objectKey string) error {
 _, err := basics.S3Client.CopyObject(ctx, &s3.CopyObjectInput{
  Bucket:     aws.String(destinationBucket),
  CopySource: aws.String(fmt.Sprintf("%v/%v", sourceBucket, objectKey)),
  Key:        aws.String(objectKey),
 })
 if err != nil {
  var notActive *types.ObjectNotInActiveTierError
  if errors.As(err, &notActive) {
   log.Printf("Couldn't copy object %s from %s because the object isn't in the
 active tier.\n",
    objectKey, sourceBucket)
   err = notActive
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(destinationBucket), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }
 return err
}
```

- For API details, see CopyObject in *AWS SDK for Go API Reference*.

`CreateBucket`

The following code example shows how to use `CreateBucket`.

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.

Create a bucket with default configuration.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// CreateBucket creates a bucket with the specified name in the specified Region.
func (basics BucketBasics) CreateBucket(ctx context.Context, name string, region
 string) error {
 _, err := basics.S3Client.CreateBucket(ctx, &s3.CreateBucketInput{
  Bucket: aws.String(name),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 })
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", name)
```

```
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", name)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(name)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", name)
  }
 }
 return err
}
```

Create a bucket with object locking and wait for it to exist.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}
```

```
// CreateBucketWithLock creates a new S3 bucket with optional object locking enabled
// and waits for the bucket to exist before returning.
func (actor S3Actions) CreateBucketWithLock(ctx context.Context, bucket string,
 region string, enableObjectLock bool) (string, error) {
 input := &s3.CreateBucketInput{
  Bucket: aws.String(bucket),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 }

 if enableObjectLock {
  input.ObjectLockEnabledForBucket = aws.Bool(true)
 }

 _, err := actor.S3Client.CreateBucket(ctx, input)
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", bucket)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", bucket)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", bucket)
  }
 }

 return bucket, err
}
```

- For API details, see CreateBucket in *AWS SDK for Go API Reference*.

`DeleteBucket`

The following code example shows how to use `DeleteBucket`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// DeleteBucket deletes a bucket. The bucket must be empty or an error is returned.
func (basics BucketBasics) DeleteBucket(ctx context.Context, bucketName string)
 error {
```

```
 _, err := basics.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
  Bucket: aws.String(bucketName)})
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucketName)
   err = noBucket
  } else {
   log.Printf("Couldn't delete bucket %v. Here's why: %v\n", bucketName, err)
  }
 } else {
  err = s3.NewBucketNotExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucketName)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucketName)
  } else {
   log.Printf("Deleted %s.\n", bucketName)
  }
 }
 return err
}
```

- For API details, see DeleteBucket in *AWS SDK for Go API Reference*.

`DeleteObject`

The following code example shows how to use `DeleteObject`.

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 (
 "bytes"
 "context"
```

```
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// DeleteObject deletes an object from a bucket.
func (actor S3Actions) DeleteObject(ctx context.Context, bucket string, key string,
 versionId string, bypassGovernance bool) (bool, error) {
 deleted := false
 input := &s3.DeleteObjectInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 _, err := actor.S3Client.DeleteObject(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "AccessDenied":
```

```
    log.Printf("Access denied: cannot delete object %s from %s.\n", key, bucket)
    err = nil
   case "InvalidArgument":
    if bypassGovernance {
     log.Printf("You cannot specify bypass governance on a bucket without lock
 enabled.")
     err = nil
    }
   }
  }
 } else {
  err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: aws.String(key)},
 time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s in bucket %s to be deleted.\n",
 key, bucket)
  } else {
   deleted = true
  }
 }
 return deleted, err
}
```

- For API details, see DeleteObject in *AWS SDK for Go API Reference*.

`DeleteObjects`

The following code example shows how to use `DeleteObjects`.

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 (
```

```
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// DeleteObjects deletes a list of objects from a bucket.
func (actor S3Actions) DeleteObjects(ctx context.Context, bucket string, objects
 []types.ObjectIdentifier, bypassGovernance bool) error {
 if len(objects) == 0 {
  return nil
 }

 input := s3.DeleteObjectsInput{
  Bucket: aws.String(bucket),
  Delete: &types.Delete{
   Objects: objects,
   Quiet:   aws.Bool(true),
  },
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 delOut, err := actor.S3Client.DeleteObjects(ctx, &input)
 if err != nil || len(delOut.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucket)
  if err != nil {
   var noBucket *types.NoSuchBucket
```

```
    if errors.As(err, &noBucket) {
     log.Printf("Bucket %s does not exist.\n", bucket)
     err = noBucket
    }
   } else if len(delOut.Errors) > 0 {
    for _, outErr := range delOut.Errors {
     log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
    }
    err = fmt.Errorf("%s", *delOut.Errors[0].Message)
   }
  } else {
   for _, delObjs := range delOut.Deleted {
    err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
     ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: delObjs.Key},
  time.Minute)
    if err != nil {
     log.Printf("Failed attempt to wait for object %s to be deleted.\n",
  *delObjs.Key)
    } else {
     log.Printf("Deleted %s.\n", *delObjs.Key)
    }
   }
  }
  return err
}
```

- For API details, see DeleteObjects in *AWS SDK for Go API Reference*.

`GetObject`

The following code example shows how to use `GetObject`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// DownloadFile gets an object from a bucket and stores it in a local file.
func (basics BucketBasics) DownloadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 result, err := basics.S3Client.GetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Can't get object %s from bucket %s. No such key exists.\n",
 objectKey, bucketName)
   err = noKey
  } else {
```

```
    log.Printf("Couldn't get object %v:%v. Here's why: %v\n", bucketName, objectKey,
 err)
  }
  return err
 }
 defer result.Body.Close()
 file, err := os.Create(fileName)
 if err != nil {
  log.Printf("Couldn't create file %v. Here's why: %v\n", fileName, err)
  return err
 }
 defer file.Close()
 body, err := io.ReadAll(result.Body)
 if err != nil {
  log.Printf("Couldn't read object body from %v. Here's why: %v\n", objectKey, err)
 }
 _, err = file.Write(body)
 return err
}
```

- For API details, see GetObject in *AWS SDK for Go API Reference*.

`GetObjectLegalHold`

The following code example shows how to use `GetObjectLegalHold`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
```

```
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectLegalHold retrieves the legal hold status for an S3 object.
func (actor S3Actions) GetObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string) (*types.ObjectLockLegalHoldStatus, error) {
 var status *types.ObjectLockLegalHoldStatus
 input := &s3.GetObjectLegalHoldInput{
  Bucket:    aws.String(bucket),
  Key:       aws.String(key),
  VersionId: aws.String(versionId),
 }

 output, err := actor.S3Client.GetObjectLegalHold(ctx, input)
 if err != nil {
  var noSuchKeyErr *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noSuchKeyErr) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noSuchKeyErr
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    log.Printf("Object %s does not have an object lock configuration.\n", key)
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
    err = nil
   }
```

```
  }
 } else {
  status = &output.LegalHold.Status
 }

 return status, err
}
```

- For API details, see GetObjectLegalHold in *AWS SDK for Go API Reference*.

`GetObjectLockConfiguration`

The following code example shows how to use `GetObjectLockConfiguration`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
```

```
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectLockConfiguration retrieves the object lock configuration for an S3
 bucket.
func (actor S3Actions) GetObjectLockConfiguration(ctx context.Context, bucket
 string) (*types.ObjectLockConfiguration, error) {
 var lockConfig *types.ObjectLockConfiguration
 input := &s3.GetObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
 }

 output, err := actor.S3Client.GetObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  } else if errors.As(err, &apiErr) && apiErr.ErrorCode() ==
 "ObjectLockConfigurationNotFoundError" {
   log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
   err = nil
  }
 } else {
  lockConfig = output.ObjectLockConfiguration
 }

 return lockConfig, err
}
```

- For API details, see GetObjectLockConfiguration in *AWS SDK for Go API Reference*.

`GetObjectRetention`

The following code example shows how to use `GetObjectRetention`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// GetObjectRetention retrieves the object retention configuration for an S3 object.
func (actor S3Actions) GetObjectRetention(ctx context.Context, bucket string, key
 string) (*types.ObjectLockRetention, error) {
 var retention *types.ObjectLockRetention
 input := &s3.GetObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }

 output, err := actor.S3Client.GetObjectRetention(ctx, input)
```

```
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have locking enabled.", bucket)
    err = nil
   }
  }
 } else {
  retention = output.Retention
 }

 return retention, err
}
```

- For API details, see GetObjectRetention in *AWS SDK for Go API Reference*.

`HeadBucket`

The following code example shows how to use `HeadBucket`.

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 (
 "bytes"
 "context"
```

```
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// BucketExists checks whether a bucket exists in the current account.
func (basics BucketBasics) BucketExists(ctx context.Context, bucketName string)
 (bool, error) {
 _, err := basics.S3Client.HeadBucket(ctx, &s3.HeadBucketInput{
  Bucket: aws.String(bucketName),
 })
 exists := true
 if err != nil {
  var apiError smithy.APIError
  if errors.As(err, &apiError) {
   switch apiError.(type) {
   case *types.NotFound:
    log.Printf("Bucket %v is available.\n", bucketName)
    exists = false
    err = nil
   default:
    log.Printf("Either you don't have access to bucket %v or another error occurred.
 "+
     "Here's what happened: %v\n", bucketName, err)
   }
```

```
  }
 } else {
  log.Printf("Bucket %v exists and you already own it.", bucketName)
 }

 return exists, err
}
```

- For API details, see HeadBucket in *AWS SDK for Go API Reference*.

`ListBuckets`

The following code example shows how to use `ListBuckets`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)
```

```
// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListBuckets lists the buckets in the current account.
func (basics BucketBasics) ListBuckets(ctx context.Context) ([]types.Bucket, error)
 {
 var err error
 var output *s3.ListBucketsOutput
 var buckets []types.Bucket
 bucketPaginator := s3.NewListBucketsPaginator(basics.S3Client,
 &s3.ListBucketsInput{})
 for bucketPaginator.HasMorePages() {
  output, err = bucketPaginator.NextPage(ctx)
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "AccessDenied" {
    fmt.Println("You don't have permission to list buckets for this account.")
    err = apiErr
   } else {
    log.Printf("Couldn't list buckets for your account. Here's why: %v\n", err)
   }
   break
  } else {
   buckets = append(buckets, output.Buckets...)
  }
 }
 return buckets, err
}
```

- For API details, see ListBuckets in *AWS SDK for Go API Reference*.

`ListObjectVersions`

The following code example shows how to use `ListObjectVersions`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// ListObjectVersions lists all versions of all objects in a bucket.
func (actor S3Actions) ListObjectVersions(ctx context.Context, bucket string)
 ([]types.ObjectVersion, error) {
 var err error
 var output *s3.ListObjectVersionsOutput
 var versions []types.ObjectVersion
 input := &s3.ListObjectVersionsInput{Bucket: aws.String(bucket)}
 versionPaginator := s3.NewListObjectVersionsPaginator(actor.S3Client, input)
 for versionPaginator.HasMorePages() {
  output, err = versionPaginator.NextPage(ctx)
```

```
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
   break
  } else {
   versions = append(versions, output.Versions...)
  }
 }
 return versions, err
}
```

- For API details, see ListObjectVersions in *AWS SDK for Go API Reference*.

`ListObjectsV2`

The following code example shows how to use `ListObjectsV2`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
```

```
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// ListObjects lists the objects in a bucket.
func (basics BucketBasics) ListObjects(ctx context.Context, bucketName string)
 ([]types.Object, error) {
 var err error
 var output *s3.ListObjectsV2Output
 input := &s3.ListObjectsV2Input{
  Bucket: aws.String(bucketName),
 }
 var objects []types.Object
 objectPaginator := s3.NewListObjectsV2Paginator(basics.S3Client, input)
 for objectPaginator.HasMorePages() {
  output, err = objectPaginator.NextPage(ctx)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucketName)
    err = noBucket
   }
   break
  } else {
   objects = append(objects, output.Contents...)
  }
 }
 return objects, err
}
```

- For API details, see ListObjectsV2 in *AWS SDK for Go API Reference*.

`PutObject`

The following code example shows how to use `PutObject`.

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.

Put an object in a bucket by using the low-level API.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}
```

```
// UploadFile reads from a file and puts the data into an object in a bucket.
func (basics BucketBasics) UploadFile(ctx context.Context, bucketName string,
 objectKey string, fileName string) error {
 file, err := os.Open(fileName)
 if err != nil {
  log.Printf("Couldn't open file %v to upload. Here's why: %v\n", fileName, err)
 } else {
  defer file.Close()
  _, err = basics.S3Client.PutObject(ctx, &s3.PutObjectInput{
   Bucket: aws.String(bucketName),
   Key:    aws.String(objectKey),
   Body:   file,
  })
  if err != nil {
   var apiErr smithy.APIError
   if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
    log.Printf("Error while uploading object to %s. The object is too large.\n"+
     "To upload objects larger than 5GB, use the S3 console (160GB max)\n"+
     "or the multipart upload API (5TB max).", bucketName)
   } else {
    log.Printf("Couldn't upload file %v to %v:%v. Here's why: %v\n",
     fileName, bucketName, objectKey, err)
   }
  } else {
   err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
   }
  }
 }
 return err
}
```

Upload an object to a bucket by using a transfer manager.

```
import (
```

```
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// UploadObject uses the S3 upload manager to upload an object to a bucket.
func (actor S3Actions) UploadObject(ctx context.Context, bucket string, key string,
 contents string) (string, error) {
 var outKey string
 input := &s3.PutObjectInput{
  Bucket:            aws.String(bucket),
  Key:               aws.String(key),
  Body:              bytes.NewReader([]byte(contents)),
  ChecksumAlgorithm: types.ChecksumAlgorithmSha256,
 }
 output, err := actor.S3Manager.Upload(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 } else {
  err := s3.NewObjectExistsWaiter(actor.S3Client).Wait(ctx, &s3.HeadObjectInput{
   Bucket: aws.String(bucket),
   Key:    aws.String(key),
  }, time.Minute)
```

```
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist in %s.\n", key, bucket)
  } else {
   outKey = *output.Key
  }
 }
 return outKey, err
}
```

- For API details, see PutObject in *AWS SDK for Go API Reference*.

`PutObjectLegalHold`

The following code example shows how to use `PutObjectLegalHold`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)
```

```
// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// PutObjectLegalHold sets the legal hold configuration for an S3 object.
func (actor S3Actions) PutObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string, legalHoldStatus types.ObjectLockLegalHoldStatus) error {
 input := &s3.PutObjectLegalHoldInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  LegalHold: &types.ObjectLockLegalHold{
   Status: legalHoldStatus,
  },
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }

 _, err := actor.S3Client.PutObjectLegalHold(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}
```

- For API details, see PutObjectLegalHold in *AWS SDK for Go API Reference*.

`PutObjectLockConfiguration`

The following code example shows how to use `PutObjectLockConfiguration`.

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.

Set the object lock configuration of a bucket.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// EnableObjectLockOnBucket enables object locking on an existing bucket.
func (actor S3Actions) EnableObjectLockOnBucket(ctx context.Context, bucket string)
 error {
 // Versioning must be enabled on the bucket before object locking is enabled.
 verInput := &s3.PutBucketVersioningInput{
  Bucket: aws.String(bucket),
  VersioningConfiguration: &types.VersioningConfiguration{
   MFADelete: types.MFADeleteDisabled,
   Status:    types.BucketVersioningStatusEnabled,
```

```
  },
 }
 _, err := actor.S3Client.PutBucketVersioning(ctx, verInput)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
  return err
 }

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: types.ObjectLockEnabledEnabled,
  },
 }
 _, err = actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }

 return err
}
```

Set the default retention period of a bucket.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// ModifyDefaultBucketRetention modifies the default retention period of an existing
 bucket.
func (actor S3Actions) ModifyDefaultBucketRetention(
 ctx context.Context, bucket string, lockMode types.ObjectLockEnabled,
 retentionPeriod int32, retentionMode types.ObjectLockRetentionMode) error {

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: lockMode,
   Rule: &types.ObjectLockRule{
    DefaultRetention: &types.DefaultRetention{
     Days: aws.Int32(retentionPeriod),
     Mode: retentionMode,
    },
   },
  },
 }
 _, err := actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }

 return err
}
```

- For API details, see PutObjectLockConfiguration in *AWS SDK for Go API Reference*.

`PutObjectRetention`

The following code example shows how to use `PutObjectRetention`.

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 (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}
```

```
// PutObjectRetention sets the object retention configuration for an S3 object.
func (actor S3Actions) PutObjectRetention(ctx context.Context, bucket string, key
 string, retentionMode types.ObjectLockRetentionMode, retentionPeriodDays int32)
 error {
 input := &s3.PutObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  Retention: &types.ObjectLockRetention{
   Mode:            retentionMode,
   RetainUntilDate: aws.Time(time.Now().AddDate(0, 0, int(retentionPeriodDays))),
  },
  BypassGovernanceRetention: aws.Bool(true),
 }

 _, err := actor.S3Client.PutObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}
```

- For API details, see PutObjectRetention in *AWS SDK for Go API Reference*.

Scenarios

Create a presigned URL

The following code example shows how to create a presigned URL for Amazon S3 and upload an object.

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.

Create functions that wrap S3 presigning actions.

```
import (
 "context"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 v4 "github.com/aws/aws-sdk-go-v2/aws/signer/v4"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

// Presigner encapsulates the Amazon Simple Storage Service (Amazon S3) presign
 actions
// used in the examples.
// It contains PresignClient, a client that is used to presign requests to Amazon
 S3.
// Presigned requests contain temporary credentials and can be made from any HTTP
 client.
type Presigner struct {
 PresignClient *s3.PresignClient
}



// GetObject makes a presigned request that can be used to get an object from a
 bucket.
// The presigned request is valid for the specified number of seconds.
func (presigner Presigner) GetObject(
 ctx context.Context, bucketName string, objectKey string, lifetimeSecs int64)
 (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignGetObject(ctx, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
```

```
 }, func(opts *s3.PresignOptions) {
  opts.Expires = time.Duration(lifetimeSecs * int64(time.Second))
 })
 if err != nil {
  log.Printf("Couldn't get a presigned request to get %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return request, err
}



// PutObject makes a presigned request that can be used to put an object in a
 bucket.
// The presigned request is valid for the specified number of seconds.
func (presigner Presigner) PutObject(
 ctx context.Context, bucketName string, objectKey string, lifetimeSecs int64)
 (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignPutObject(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 }, func(opts *s3.PresignOptions) {
  opts.Expires = time.Duration(lifetimeSecs * int64(time.Second))
 })
 if err != nil {
  log.Printf("Couldn't get a presigned request to put %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return request, err
}



// DeleteObject makes a presigned request that can be used to delete an object from
 a bucket.
func (presigner Presigner) DeleteObject(ctx context.Context, bucketName string,
 objectKey string) (*v4.PresignedHTTPRequest, error) {
 request, err := presigner.PresignClient.PresignDeleteObject(ctx,
 &s3.DeleteObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
```

```
  log.Printf("Couldn't get a presigned request to delete object %v. Here's why: %v
\n", objectKey, err)
 }
 return request, err
}



func (presigner Presigner) PresignPostObject(ctx context.Context, bucketName string,
 objectKey string, lifetimeSecs int64) (*s3.PresignedPostRequest, error) {
 request, err := presigner.PresignClient.PresignPostObject(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 }, func(options *s3.PresignPostOptions) {
  options.Expires = time.Duration(lifetimeSecs) * time.Second
 })
 if err != nil {
  log.Printf("Couldn't get a presigned post request to put %v:%v. Here's why: %v\n",
 bucketName, objectKey, err)
 }
 return request, nil
}
```

Run an interactive example that generates and uses presigned URLs to upload, download, and delete an S3 object.

```
import (
 "bytes"
 "context"
 "io"
 "log"
 "mime/multipart"
 "net/http"
 "os"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
```

```
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)



// RunPresigningScenario is an interactive example that shows you how to get
 presigned
// HTTP requests that you can use to move data into and out of Amazon Simple Storage
// Service (Amazon S3). The presigned requests contain temporary credentials and can
// be used by an HTTP client.
//
// 1. Get a presigned request to put an object in a bucket.
// 2. Use the net/http package to use the presigned request to upload a local file
 to the bucket.
// 3. Get a presigned request to get an object from a bucket.
// 4. Use the net/http package to use the presigned request to download the object
 to a local file.
// 5. Get a presigned request to delete an object from a bucket.
// 6. Use the net/http package to use the presigned request to delete the object.
//
// This example creates an Amazon S3 presign client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
//
// It uses an IHttpRequester interface to abstract HTTP requests so they can be
 mocked
// during testing.
func RunPresigningScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner, httpRequester IHttpRequester) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
  }
 }()
```

```
log.Println(strings.Repeat("-", 88))
log.Println("Welcome to the Amazon S3 presigning demo.")
log.Println(strings.Repeat("-", 88))

s3Client := s3.NewFromConfig(sdkConfig)
bucketBasics := actions.BucketBasics{S3Client: s3Client}
presignClient := s3.NewPresignClient(s3Client)
presigner := actions.Presigner{PresignClient: presignClient}

bucketName := questioner.Ask("We'll need a bucket. Enter a name for a bucket "+
 "you own or one you want to create:", demotools.NotEmpty{})
bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
if err != nil {
 panic(err)
}
if !bucketExists {
 err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
 if err != nil {
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))

log.Printf("Let's presign a request to upload a file to your bucket.")
uploadFilename := questioner.Ask("Enter the path to a file you want to upload:",
 demotools.NotEmpty{})
uploadKey := questioner.Ask("What would you like to name the uploaded object?",
 demotools.NotEmpty{})
uploadFile, err := os.Open(uploadFilename)
if err != nil {
 panic(err)
}
defer uploadFile.Close()
presignedPutRequest, err := presigner.PutObject(ctx, bucketName, uploadKey, 60)
if err != nil {
 panic(err)
}
log.Printf("Got a presigned %v request to URL:\n\t%v\n",
presignedPutRequest.Method,
 presignedPutRequest.URL)
log.Println("Using net/http to send the request...")
info, err := uploadFile.Stat()
```

```
if err != nil {
 panic(err)
}
putResponse, err := httpRequester.Put(presignedPutRequest.URL, info.Size(),
uploadFile)
if err != nil {
 panic(err)
}
log.Printf("%v object %v with presigned URL returned %v.",
presignedPutRequest.Method,
 uploadKey, putResponse.StatusCode)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's presign a request to download the object.")
questioner.Ask("Press Enter when you're ready.")
presignedGetRequest, err := presigner.GetObject(ctx, bucketName, uploadKey, 60)
if err != nil {
 panic(err)
}
log.Printf("Got a presigned %v request to URL:\n\t%v\n",
presignedGetRequest.Method,
 presignedGetRequest.URL)
log.Println("Using net/http to send the request...")
getResponse, err := httpRequester.Get(presignedGetRequest.URL)
if err != nil {
 panic(err)
}
log.Printf("%v object %v with presigned URL returned %v.",
presignedGetRequest.Method,
 uploadKey, getResponse.StatusCode)
defer getResponse.Body.Close()
downloadBody, err := io.ReadAll(getResponse.Body)
if err != nil {
 panic(err)
}
log.Printf("Downloaded %v bytes. Here are the first 100 of them:\n",
len(downloadBody))
log.Println(strings.Repeat("-", 88))
log.Println(string(downloadBody[:100]))
log.Println(strings.Repeat("-", 88))

log.Println("Now we'll create a new request to put the same object using a
presigned post request")
questioner.Ask("Press Enter when you're ready.")
```

```
 presignPostRequest, err := presigner.PresignPostObject(ctx, bucketName, uploadKey,
 60)
 if err != nil {
  panic(err)
 }
 log.Printf("Got a presigned post request to url %v with values %v\n",
 presignPostRequest.URL, presignPostRequest.Values)
 log.Println("Using net/http multipart to send the request...")
 uploadFile, err = os.Open(uploadFilename)
 if err != nil {
  panic(err)
 }
 defer uploadFile.Close()
 multiPartResponse, err := sendMultipartRequest(presignPostRequest.URL,
 presignPostRequest.Values, uploadFile, uploadKey, httpRequester)
 if err != nil {
  panic(err)
 }
 log.Printf("Presign post object %v with presigned URL returned %v.", uploadKey,
 multiPartResponse.StatusCode)

 log.Println("Let's presign a request to delete the object.")
 questioner.Ask("Press Enter when you're ready.")
 presignedDelRequest, err := presigner.DeleteObject(ctx, bucketName, uploadKey)
 if err != nil {
  panic(err)
 }
 log.Printf("Got a presigned %v request to URL:\n\t%v\n",
 presignedDelRequest.Method,
  presignedDelRequest.URL)
 log.Println("Using net/http to send the request...")
 delResponse, err := httpRequester.Delete(presignedDelRequest.URL)
 if err != nil {
  panic(err)
 }
 log.Printf("%v object %v with presigned URL returned %v.\n",
 presignedDelRequest.Method,
  uploadKey, delResponse.StatusCode)
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define an HTTP request wrapper used by the example to make HTTP requests.

```
// IHttpRequester abstracts HTTP requests into an interface so it can be mocked
 during
// unit testing.
type IHttpRequester interface {
 Get(url string) (resp *http.Response, err error)
 Post(url, contentType string, body io.Reader) (resp *http.Response, err error)
 Put(url string, contentLength int64, body io.Reader) (resp *http.Response, err
 error)
 Delete(url string) (resp *http.Response, err error)
}

// HttpRequester uses the net/http package to make HTTP requests during the
 scenario.
type HttpRequester struct{}

func (httpReq HttpRequester) Get(url string) (resp *http.Response, err error) {
 return http.Get(url)
}
func (httpReq HttpRequester) Post(url, contentType string, body io.Reader) (resp
 *http.Response, err error) {
 postRequest, err := http.NewRequest("POST", url, body)
 if err != nil {
  return nil, err
 }
 postRequest.Header.Set("Content-Type", contentType)
 return http.DefaultClient.Do(postRequest)
}

func (httpReq HttpRequester) Put(url string, contentLength int64, body io.Reader)
 (resp *http.Response, err error) {
 putRequest, err := http.NewRequest("PUT", url, body)
 if err != nil {
  return nil, err
 }
 putRequest.ContentLength = contentLength
 return http.DefaultClient.Do(putRequest)
}
func (httpReq HttpRequester) Delete(url string) (resp *http.Response, err error) {
```

```
 delRequest, err := http.NewRequest("DELETE", url, nil)
 if err != nil {
  return nil, err
 }
 return http.DefaultClient.Do(delRequest)
}
```

Lock Amazon S3 objects

The following code example shows how to work with S3 object lock features.

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 demonstrating Amazon S3 object lock features.

```
import (
 "context"
 "fmt"
 "log"
 "strings"

 "s3_object_lock/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// ObjectLockScenario contains the steps to run the S3 Object Lock workflow.
type ObjectLockScenario struct {
 questioner demotools.IQuestioner
```

```
 resources  Resources
 s3Actions  *actions.S3Actions
 sdkConfig  aws.Config
}

// NewObjectLockScenario constructs a new ObjectLockScenario instance.
func NewObjectLockScenario(sdkConfig aws.Config, questioner demotools.IQuestioner)
 ObjectLockScenario {
 scenario := ObjectLockScenario{
  questioner: questioner,
  resources:  Resources{},
  s3Actions:  &actions.S3Actions{S3Client: s3.NewFromConfig(sdkConfig)},
  sdkConfig:  sdkConfig,
 }
 scenario.s3Actions.S3Manager = manager.NewUploader(scenario.s3Actions.S3Client)
 scenario.resources.init(scenario.s3Actions, questioner)
 return scenario
}

type nameLocked struct {
 name   string
 locked bool
}

var createInfo = []nameLocked{
 {"standard-bucket", false},
 {"lock-bucket", true},
 {"retention-bucket", false},
}

// CreateBuckets creates the S3 buckets required for the workflow.
func (scenario *ObjectLockScenario) CreateBuckets(ctx context.Context) {
 log.Println("Let's create some S3 buckets to use for this workflow.")
 success := false
 for !success {
  prefix := scenario.questioner.Ask(
   "This example creates three buckets. Enter a prefix to name your buckets
 (remember bucket names must be globally unique):")

  for _, info := range createInfo {
   log.Println(fmt.Sprintf("%s.%s", prefix, info.name))
   bucketName, err := scenario.s3Actions.CreateBucketWithLock(ctx, fmt.Sprintf("%s.
%s", prefix, info.name), scenario.sdkConfig.Region, info.locked)
   if err != nil {
```

```
    switch err.(type) {
    case *types.BucketAlreadyExists, *types.BucketAlreadyOwnedByYou:
     log.Printf("Couldn't create bucket %s.\n", bucketName)
    default:
     panic(err)
    }
    break
   }
   scenario.resources.demoBuckets[info.name] = &DemoBucket{
    name:       bucketName,
    objectKeys: []string{},
   }
   log.Printf("Created bucket %s.\n", bucketName)
  }

  if len(scenario.resources.demoBuckets) < len(createInfo) {
   scenario.resources.deleteBuckets(ctx)
  } else {
   success = true
  }
 }

 log.Println("S3 buckets created.")
 log.Println(strings.Repeat("-", 88))
}

// EnableLockOnBucket enables object locking on an existing bucket.
func (scenario *ObjectLockScenario) EnableLockOnBucket(ctx context.Context) {
 log.Println("\nA bucket can be configured to use object locking.")
 scenario.questioner.Ask("Press Enter to continue.")

 var err error
 bucket := scenario.resources.demoBuckets["retention-bucket"]
 err = scenario.s3Actions.EnableObjectLockOnBucket(ctx, bucket.name)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("Couldn't enable object locking on bucket %s.\n", bucket.name)
  default:
   panic(err)
  }
 } else {
  log.Printf("Object locking enabled on bucket %s.", bucket.name)
 }
```

```
 log.Println(strings.Repeat("-", 88))
}

// SetDefaultRetentionPolicy sets a default retention governance policy on a bucket.
func (scenario *ObjectLockScenario) SetDefaultRetentionPolicy(ctx context.Context) {
 log.Println("\nA bucket can be configured to use object locking with a default
 retention period.")

 bucket := scenario.resources.demoBuckets["retention-bucket"]
 retentionPeriod := scenario.questioner.AskInt("Enter the default retention period
 in days: ")
 err := scenario.s3Actions.ModifyDefaultBucketRetention(ctx,
 bucket.name, types.ObjectLockEnabledEnabled, int32(retentionPeriod),
 types.ObjectLockRetentionModeGovernance)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("Couldn't configure a default retention period on bucket %s.\n",
 bucket.name)
  default:
   panic(err)
  }
 } else {
  log.Printf("Default retention policy set on bucket %s with %d day retention
 period.", bucket.name, retentionPeriod)
  bucket.retentionEnabled = true
 }

 log.Println(strings.Repeat("-", 88))
}

// UploadTestObjects uploads test objects to the S3 buckets.
func (scenario *ObjectLockScenario) UploadTestObjects(ctx context.Context) {
 log.Println("Uploading test objects to S3 buckets.")

 for _, info := range createInfo {
  bucket := scenario.resources.demoBuckets[info.name]
  for i := 0; i < 2; i++ {
   key, err := scenario.s3Actions.UploadObject(ctx, bucket.name,
 fmt.Sprintf("example-%d", i),
    fmt.Sprintf("Example object content #%d in bucket %s.", i, bucket.name))
   if err != nil {
    switch err.(type) {
```

```
    case *types.NoSuchBucket:
     log.Printf("Couldn't upload %s to bucket %s.\n", key, bucket.name)
    default:
     panic(err)
    }
   } else {
    log.Printf("Uploaded %s to bucket %s.\n", key, bucket.name)
    bucket.objectKeys = append(bucket.objectKeys, key)
   }
  }
 }

 scenario.questioner.Ask("Test objects uploaded. Press Enter to continue.")
 log.Println(strings.Repeat("-", 88))
}

// SetObjectLockConfigurations sets object lock configurations on the test objects.
func (scenario *ObjectLockScenario) SetObjectLockConfigurations(ctx context.Context)
 {
 log.Println("Now let's set object lock configurations on individual objects.")

 buckets := []*DemoBucket{scenario.resources.demoBuckets["lock-bucket"],
 scenario.resources.demoBuckets["retention-bucket"]}
 for _, bucket := range buckets {
  for index, objKey := range bucket.objectKeys {
   switch index {
   case 0:
    if scenario.questioner.AskBool(fmt.Sprintf("\nDo you want to add a legal hold to
 %s in %s (y/n)? ", objKey, bucket.name), "y") {
     err := scenario.s3Actions.PutObjectLegalHold(ctx, bucket.name, objKey, "",
 types.ObjectLockLegalHoldStatusOn)
     if err != nil {
      switch err.(type) {
      case *types.NoSuchKey:
       log.Printf("Couldn't set legal hold on %s.\n", objKey)
      default:
       panic(err)
      }
     } else {
      log.Printf("Legal hold set on %s.\n", objKey)
     }
    }
   case 1:
```

```
    q := fmt.Sprintf("\nDo you want to add a 1 day Governance retention period to %s
 in %s?\n"+
     "Reminder: Only a user with the s3:BypassGovernanceRetention permission is able
 to delete this object\n"+
     "or its bucket until the retention period has expired. (y/n) ", objKey,
 bucket.name)
    if scenario.questioner.AskBool(q, "y") {
     err := scenario.s3Actions.PutObjectRetention(ctx, bucket.name, objKey,
 types.ObjectLockRetentionModeGovernance, 1)
     if err != nil {
      switch err.(type) {
      case *types.NoSuchKey:
       log.Printf("Couldn't set retention period on %s in %s.\n", objKey,
 bucket.name)
      default:
       panic(err)
      }
     } else {
      log.Printf("Retention period set to 1 for %s.", objKey)
      bucket.retentionEnabled = true
     }
    }
   }
  }
 }
 log.Println(strings.Repeat("-", 88))
}

const (
 ListAll = iota
 DeleteObject
 DeleteRetentionObject
 OverwriteObject
 ViewRetention
 ViewLegalHold
 Finish
)

// InteractWithObjects allows the user to interact with the objects and test the
 object lock configurations.
func (scenario *ObjectLockScenario) InteractWithObjects(ctx context.Context) {
 log.Println("Now you can interact with the objects to explore the object lock
 configurations.")
 interactiveChoices := []string{
```

```
 "List all objects and buckets.",
 "Attempt to delete an object.",
 "Attempt to delete an object with retention period bypass.",
 "Attempt to overwrite a file.",
 "View the retention settings for an object.",
 "View the legal hold settings for an object.",
 "Finish the workflow."}

choice := ListAll
for choice != Finish {
 objList := scenario.GetAllObjects(ctx)
 objChoices := scenario.makeObjectChoiceList(objList)
 choice = scenario.questioner.AskChoice("Choose an action from the menu:\n",
interactiveChoices)
 switch choice {
 case ListAll:
  log.Println("The current objects in the example buckets are:")
  for _, objChoice := range objChoices {
   log.Println("\t", objChoice)
  }
 case DeleteObject, DeleteRetentionObject:
  objChoice := scenario.questioner.AskChoice("Enter the number of the object to
delete:\n", objChoices)
  obj := objList[objChoice]
  deleted, err := scenario.s3Actions.DeleteObject(ctx, obj.bucket, obj.key,
obj.versionId, choice == DeleteRetentionObject)
  if err != nil {
   switch err.(type) {
   case *types.NoSuchKey:
    log.Println("Nothing to delete.")
   default:
    panic(err)
   }
  } else if deleted {
   log.Printf("Object %s deleted.\n", obj.key)
  }
 case OverwriteObject:
  objChoice := scenario.questioner.AskChoice("Enter the number of the object to
overwrite:\n", objChoices)
  obj := objList[objChoice]
  _, err := scenario.s3Actions.UploadObject(ctx, obj.bucket, obj.key,
fmt.Sprintf("New content in object %s.", obj.key))
  if err != nil {
   switch err.(type) {
```

```
    case *types.NoSuchBucket:
     log.Println("Couldn't upload to nonexistent bucket.")
    default:
     panic(err)
    }
   } else {
    log.Printf("Uploaded new content to object %s.\n", obj.key)
   }
  case ViewRetention:
   objChoice := scenario.questioner.AskChoice("Enter the number of the object to
view:\n", objChoices)
   obj := objList[objChoice]
   retention, err := scenario.s3Actions.GetObjectRetention(ctx, obj.bucket, obj.key)
   if err != nil {
    switch err.(type) {
    case *types.NoSuchKey:
     log.Printf("Can't get retention configuration for %s.\n", obj.key)
    default:
     panic(err)
    }
   } else if retention != nil {
    log.Printf("Object %s has retention mode %s until %v.\n", obj.key,
retention.Mode, retention.RetainUntilDate)
   } else {
    log.Printf("Object %s does not have object retention configured.\n", obj.key)
   }
  case ViewLegalHold:
   objChoice := scenario.questioner.AskChoice("Enter the number of the object to
view:\n", objChoices)
   obj := objList[objChoice]
   legalHold, err := scenario.s3Actions.GetObjectLegalHold(ctx, obj.bucket, obj.key,
obj.versionId)
   if err != nil {
    switch err.(type) {
    case *types.NoSuchKey:
     log.Printf("Can't get legal hold configuration for %s.\n", obj.key)
    default:
     panic(err)
    }
   } else if legalHold != nil {
    log.Printf("Object %s has legal hold %v.", obj.key, *legalHold)
   } else {
    log.Printf("Object %s does not have legal hold configured.", obj.key)
   }
```

```
  case Finish:
   log.Println("Let's clean up.")
  }
  log.Println(strings.Repeat("-", 88))
 }
}

type BucketKeyVersionId struct {
 bucket    string
 key       string
 versionId string
}

// GetAllObjects gets the object versions in the example S3 buckets and returns them
 in a flattened list.
func (scenario *ObjectLockScenario) GetAllObjects(ctx context.Context)
 []BucketKeyVersionId {
 var objectList []BucketKeyVersionId
 for _, info := range createInfo {
  bucket := scenario.resources.demoBuckets[info.name]
  versions, err := scenario.s3Actions.ListObjectVersions(ctx, bucket.name)
  if err != nil {
   switch err.(type) {
   case *types.NoSuchBucket:
    log.Printf("Couldn't get object versions for %s.\n", bucket.name)
   default:
    panic(err)
   }
  } else {
   for _, version := range versions {
    objectList = append(objectList,
     BucketKeyVersionId{bucket: bucket.name, key: *version.Key, versionId:
 *version.VersionId})
   }
  }
 }
 return objectList
}

// makeObjectChoiceList makes the object version list into a list of strings that
 are displayed
// as choices.
func (scenario *ObjectLockScenario) makeObjectChoiceList(bucketObjects
 []BucketKeyVersionId) []string {
```

```
 choices := make([]string, len(bucketObjects))
 for i := 0; i < len(bucketObjects); i++ {
  choices[i] = fmt.Sprintf("%s in %s with VersionId %s.",
   bucketObjects[i].key, bucketObjects[i].bucket, bucketObjects[i].versionId)
 }
 return choices
}

// Run runs the S3 Object Lock scenario.
func (scenario *ObjectLockScenario) Run(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := scenario.questioner.(*demotools.MockQuestioner)
   if isMock || scenario.questioner.AskBool("Do you want to see the full error
 message (y/n)?", "y") {
    log.Println(r)
   }
   scenario.resources.Cleanup(ctx)
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon S3 Object Lock Feature Scenario.")
 log.Println(strings.Repeat("-", 88))

 scenario.CreateBuckets(ctx)
 scenario.EnableLockOnBucket(ctx)
 scenario.SetDefaultRetentionPolicy(ctx)
 scenario.UploadTestObjects(ctx)
 scenario.SetObjectLockConfigurations(ctx)
 scenario.InteractWithObjects(ctx)

 scenario.resources.Cleanup(ctx)

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps S3 actions used in this example.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "log"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// S3Actions wraps S3 service actions.
type S3Actions struct {
 S3Client  *s3.Client
 S3Manager *manager.Uploader
}



// CreateBucketWithLock creates a new S3 bucket with optional object locking enabled
// and waits for the bucket to exist before returning.
func (actor S3Actions) CreateBucketWithLock(ctx context.Context, bucket string,
 region string, enableObjectLock bool) (string, error) {
 input := &s3.CreateBucketInput{
  Bucket: aws.String(bucket),
  CreateBucketConfiguration: &types.CreateBucketConfiguration{
   LocationConstraint: types.BucketLocationConstraint(region),
  },
 }

 if enableObjectLock {
  input.ObjectLockEnabledForBucket = aws.Bool(true)
 }

 _, err := actor.S3Client.CreateBucket(ctx, input)
 if err != nil {
  var owned *types.BucketAlreadyOwnedByYou
  var exists *types.BucketAlreadyExists
```

```
  if errors.As(err, &owned) {
   log.Printf("You already own bucket %s.\n", bucket)
   err = owned
  } else if errors.As(err, &exists) {
   log.Printf("Bucket %s already exists.\n", bucket)
   err = exists
  }
 } else {
  err = s3.NewBucketExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to exist.\n", bucket)
  }
 }

 return bucket, err
}



// GetObjectLegalHold retrieves the legal hold status for an S3 object.
func (actor S3Actions) GetObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string) (*types.ObjectLockLegalHoldStatus, error) {
 var status *types.ObjectLockLegalHoldStatus
 input := &s3.GetObjectLegalHoldInput{
  Bucket:    aws.String(bucket),
  Key:       aws.String(key),
  VersionId: aws.String(versionId),
 }

 output, err := actor.S3Client.GetObjectLegalHold(ctx, input)
 if err != nil {
  var noSuchKeyErr *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noSuchKeyErr) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noSuchKeyErr
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    log.Printf("Object %s does not have an object lock configuration.\n", key)
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
```

```
    err = nil
   }
  }
 } else {
  status = &output.LegalHold.Status
 }

 return status, err
}



// GetObjectLockConfiguration retrieves the object lock configuration for an S3
 bucket.
func (actor S3Actions) GetObjectLockConfiguration(ctx context.Context, bucket
 string) (*types.ObjectLockConfiguration, error) {
 var lockConfig *types.ObjectLockConfiguration
 input := &s3.GetObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
 }

 output, err := actor.S3Client.GetObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  } else if errors.As(err, &apiErr) && apiErr.ErrorCode() ==
 "ObjectLockConfigurationNotFoundError" {
   log.Printf("Bucket %s does not have an object lock configuration.\n", bucket)
   err = nil
  }
 } else {
  lockConfig = output.ObjectLockConfiguration
 }

 return lockConfig, err
}



// GetObjectRetention retrieves the object retention configuration for an S3 object.
```

```
func (actor S3Actions) GetObjectRetention(ctx context.Context, bucket string, key
 string) (*types.ObjectLockRetention, error) {
 var retention *types.ObjectLockRetention
 input := &s3.GetObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }

 output, err := actor.S3Client.GetObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "NoSuchObjectLockConfiguration":
    err = nil
   case "InvalidRequest":
    log.Printf("Bucket %s does not have locking enabled.", bucket)
    err = nil
   }
  }
 } else {
  retention = output.Retention
 }

 return retention, err
}



// PutObjectLegalHold sets the legal hold configuration for an S3 object.
func (actor S3Actions) PutObjectLegalHold(ctx context.Context, bucket string, key
 string, versionId string, legalHoldStatus types.ObjectLockLegalHoldStatus) error {
 input := &s3.PutObjectLegalHoldInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  LegalHold: &types.ObjectLockLegalHold{
   Status: legalHoldStatus,
  },
 }
 if versionId != "" {
```

```
  input.VersionId = aws.String(versionId)
 }

 _, err := actor.S3Client.PutObjectLegalHold(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}



// ModifyDefaultBucketRetention modifies the default retention period of an existing
 bucket.
func (actor S3Actions) ModifyDefaultBucketRetention(
 ctx context.Context, bucket string, lockMode types.ObjectLockEnabled,
 retentionPeriod int32, retentionMode types.ObjectLockRetentionMode) error {

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: lockMode,
   Rule: &types.ObjectLockRule{
    DefaultRetention: &types.DefaultRetention{
     Days: aws.Int32(retentionPeriod),
     Mode: retentionMode,
    },
   },
  },
 }
 _, err := actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }
```

```
 return err
}



// EnableObjectLockOnBucket enables object locking on an existing bucket.
func (actor S3Actions) EnableObjectLockOnBucket(ctx context.Context, bucket string)
 error {
 // Versioning must be enabled on the bucket before object locking is enabled.
 verInput := &s3.PutBucketVersioningInput{
  Bucket: aws.String(bucket),
  VersioningConfiguration: &types.VersioningConfiguration{
   MFADelete: types.MFADeleteDisabled,
   Status:    types.BucketVersioningStatusEnabled,
  },
 }
 _, err := actor.S3Client.PutBucketVersioning(ctx, verInput)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
  return err
 }

 input := &s3.PutObjectLockConfigurationInput{
  Bucket: aws.String(bucket),
  ObjectLockConfiguration: &types.ObjectLockConfiguration{
   ObjectLockEnabled: types.ObjectLockEnabledEnabled,
  },
 }
 _, err = actor.S3Client.PutObjectLockConfiguration(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 }

 return err
}
```

```
// PutObjectRetention sets the object retention configuration for an S3 object.
func (actor S3Actions) PutObjectRetention(ctx context.Context, bucket string, key
 string, retentionMode types.ObjectLockRetentionMode, retentionPeriodDays int32)
 error {
 input := &s3.PutObjectRetentionInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
  Retention: &types.ObjectLockRetention{
   Mode:            retentionMode,
   RetainUntilDate: aws.Time(time.Now().AddDate(0, 0, int(retentionPeriodDays))),
  },
  BypassGovernanceRetention: aws.Bool(true),
 }

 _, err := actor.S3Client.PutObjectRetention(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in bucket %s.\n", key, bucket)
   err = noKey
  }
 }

 return err
}



// UploadObject uses the S3 upload manager to upload an object to a bucket.
func (actor S3Actions) UploadObject(ctx context.Context, bucket string, key string,
 contents string) (string, error) {
 var outKey string
 input := &s3.PutObjectInput{
  Bucket:            aws.String(bucket),
  Key:               aws.String(key),
  Body:              bytes.NewReader([]byte(contents)),
  ChecksumAlgorithm: types.ChecksumAlgorithmSha256,
 }
 output, err := actor.S3Manager.Upload(ctx, input)
 if err != nil {
  var noBucket *types.NoSuchBucket
  if errors.As(err, &noBucket) {
```

```
   log.Printf("Bucket %s does not exist.\n", bucket)
   err = noBucket
  }
 } else {
  err := s3.NewObjectExistsWaiter(actor.S3Client).Wait(ctx, &s3.HeadObjectInput{
   Bucket: aws.String(bucket),
   Key:    aws.String(key),
  }, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist in %s.\n", key, bucket)
  } else {
   outKey = *output.Key
  }
 }
 return outKey, err
}



// ListObjectVersions lists all versions of all objects in a bucket.
func (actor S3Actions) ListObjectVersions(ctx context.Context, bucket string)
 ([]types.ObjectVersion, error) {
 var err error
 var output *s3.ListObjectVersionsOutput
 var versions []types.ObjectVersion
 input := &s3.ListObjectVersionsInput{Bucket: aws.String(bucket)}
 versionPaginator := s3.NewListObjectVersionsPaginator(actor.S3Client, input)
 for versionPaginator.HasMorePages() {
  output, err = versionPaginator.NextPage(ctx)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
   break
  } else {
   versions = append(versions, output.Versions...)
  }
 }
 return versions, err
}
```

```
// DeleteObject deletes an object from a bucket.
func (actor S3Actions) DeleteObject(ctx context.Context, bucket string, key string,
 versionId string, bypassGovernance bool) (bool, error) {
 deleted := false
 input := &s3.DeleteObjectInput{
  Bucket: aws.String(bucket),
  Key:    aws.String(key),
 }
 if versionId != "" {
  input.VersionId = aws.String(versionId)
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 _, err := actor.S3Client.DeleteObject(ctx, input)
 if err != nil {
  var noKey *types.NoSuchKey
  var apiErr *smithy.GenericAPIError
  if errors.As(err, &noKey) {
   log.Printf("Object %s does not exist in %s.\n", key, bucket)
   err = noKey
  } else if errors.As(err, &apiErr) {
   switch apiErr.ErrorCode() {
   case "AccessDenied":
    log.Printf("Access denied: cannot delete object %s from %s.\n", key, bucket)
    err = nil
   case "InvalidArgument":
    if bypassGovernance {
     log.Printf("You cannot specify bypass governance on a bucket without lock
enabled.")
     err = nil
    }
   }
  }
 } else {
  err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: aws.String(key)},
time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s in bucket %s to be deleted.\n",
key, bucket)
  } else {
   deleted = true
```

```
  }
 }
 return deleted, err
}



// DeleteObjects deletes a list of objects from a bucket.
func (actor S3Actions) DeleteObjects(ctx context.Context, bucket string, objects
 []types.ObjectIdentifier, bypassGovernance bool) error {
 if len(objects) == 0 {
  return nil
 }

 input := s3.DeleteObjectsInput{
  Bucket: aws.String(bucket),
  Delete: &types.Delete{
   Objects: objects,
   Quiet:   aws.Bool(true),
  },
 }
 if bypassGovernance {
  input.BypassGovernanceRetention = aws.Bool(true)
 }
 delOut, err := actor.S3Client.DeleteObjects(ctx, &input)
 if err != nil || len(delOut.Errors) > 0 {
  log.Printf("Error deleting objects from bucket %s.\n", bucket)
  if err != nil {
   var noBucket *types.NoSuchBucket
   if errors.As(err, &noBucket) {
    log.Printf("Bucket %s does not exist.\n", bucket)
    err = noBucket
   }
  } else if len(delOut.Errors) > 0 {
   for _, outErr := range delOut.Errors {
    log.Printf("%s: %s\n", *outErr.Key, *outErr.Message)
   }
   err = fmt.Errorf("%s", *delOut.Errors[0].Message)
  }
 } else {
  for _, delObjs := range delOut.Deleted {
   err = s3.NewObjectNotExistsWaiter(actor.S3Client).Wait(
    ctx, &s3.HeadObjectInput{Bucket: aws.String(bucket), Key: delObjs.Key},
 time.Minute)
```

```
   if err != nil {
    log.Printf("Failed attempt to wait for object %s to be deleted.\n",
 *delObjs.Key)
   } else {
    log.Printf("Deleted %s.\n", *delObjs.Key)
   }
  }
 }
 return err
}
```

Clean up resources.

```
import (
 "context"
 "log"
 "s3_object_lock/actions"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

// DemoBucket contains metadata for buckets used in this example.
type DemoBucket struct {
 name             string
 retentionEnabled bool
 objectKeys       []string
}

// Resources keeps track of AWS resources created during the ObjectLockScenario and
 handles
// cleanup when the scenario finishes.
type Resources struct {
 demoBuckets map[string]*DemoBucket

 s3Actions  *actions.S3Actions
```

```
 questioner demotools.IQuestioner
}

// init initializes objects in the Resources struct.
func (resources *Resources) init(s3Actions *actions.S3Actions, questioner
 demotools.IQuestioner) {
 resources.s3Actions = s3Actions
 resources.questioner = questioner
 resources.demoBuckets = map[string]*DemoBucket{}
}

// Cleanup deletes all AWS resources created during the ObjectLockScenario.
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 " +
    "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 {
  log.Println("Be sure to remove resources when you're done with them to avoid
 unexpected charges!")
  return
 }

 log.Println("Removing objects from S3 buckets and deleting buckets...")
 resources.deleteBuckets(ctx)
 //resources.deleteRetentionObjects(resources.retentionBucket,
 resources.retentionObjects)

 log.Println("Cleanup complete.")
}

// deleteBuckets empties and then deletes all buckets created during the
 ObjectLockScenario.
func (resources *Resources) deleteBuckets(ctx context.Context) {
 for _, info := range createInfo {
  bucket := resources.demoBuckets[info.name]
  resources.deleteObjects(ctx, bucket)
```

```
  _, err := resources.s3Actions.S3Client.DeleteBucket(ctx, &s3.DeleteBucketInput{
   Bucket: aws.String(bucket.name),
  })
  if err != nil {
   panic(err)
  }
 }
 for _, info := range createInfo {
  bucket := resources.demoBuckets[info.name]
  err := s3.NewBucketNotExistsWaiter(resources.s3Actions.S3Client).Wait(
   ctx, &s3.HeadBucketInput{Bucket: aws.String(bucket.name)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for bucket %s to be deleted.\n", bucket.name)
  } else {
   log.Printf("Deleted %s.\n", bucket.name)
  }
 }
 resources.demoBuckets = map[string]*DemoBucket{}
}

// deleteObjects deletes all objects in the specified bucket.
func (resources *Resources) deleteObjects(ctx context.Context, bucket *DemoBucket) {
 lockConfig, err := resources.s3Actions.GetObjectLockConfiguration(ctx, bucket.name)
 if err != nil {
  panic(err)
 }
 versions, err := resources.s3Actions.ListObjectVersions(ctx, bucket.name)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Printf("No objects to get from %s.\n", bucket.name)
  default:
   panic(err)
  }
 }
 delObjects := make([]types.ObjectIdentifier, len(versions))
 for i, version := range versions {
  if lockConfig != nil && lockConfig.ObjectLockEnabled ==
 types.ObjectLockEnabledEnabled {
   status, err := resources.s3Actions.GetObjectLegalHold(ctx, bucket.name,
 *version.Key, *version.VersionId)
   if err != nil {
    switch err.(type) {
    case *types.NoSuchKey:
```

```
     log.Printf("Couldn't determine legal hold status for %s in %s.\n",
 *version.Key, bucket.name)
    default:
     panic(err)
    }
   } else if status != nil && *status == types.ObjectLockLegalHoldStatusOn {
    err = resources.s3Actions.PutObjectLegalHold(ctx, bucket.name, *version.Key,
 *version.VersionId, types.ObjectLockLegalHoldStatusOff)
    if err != nil {
     switch err.(type) {
     case *types.NoSuchKey:
      log.Printf("Couldn't turn off legal hold for %s in %s.\n", *version.Key,
 bucket.name)
     default:
      panic(err)
     }
    }
   }
  }
  delObjects[i] = types.ObjectIdentifier{Key: version.Key, VersionId:
 version.VersionId}
 }
 err = resources.s3Actions.DeleteObjects(ctx, bucket.name, delObjects,
 bucket.retentionEnabled)
 if err != nil {
  switch err.(type) {
  case *types.NoSuchBucket:
   log.Println("Nothing to delete.")
  default:
   panic(err)
  }
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - GetObjectLegalHold
  - GetObjectLockConfiguration
  - GetObjectRetention
  - PutObjectLegalHold

- PutObjectLockConfiguration
- PutObjectRetention

Upload or download large files

The following code example shows how to upload or download large files to and from Amazon S3.

For more information, see Uploading an object using multipart upload.

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.

Create functions that use upload and download managers to break the data into parts and transfer them concurrently.

```
import (
 "bytes"
 "context"
 "errors"
 "fmt"
 "io"
 "log"
 "os"
 "time"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/feature/s3/manager"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/aws/aws-sdk-go-v2/service/s3/types"
 "github.com/aws/smithy-go"
)

// BucketBasics encapsulates the Amazon Simple Storage Service (Amazon S3) actions
// used in the examples.
// It contains S3Client, an Amazon S3 service client that is used to perform bucket
```

```
// and object actions.
type BucketBasics struct {
 S3Client *s3.Client
}



// UploadLargeObject uses an upload manager to upload data to an object in a bucket.
// The upload manager breaks large data into parts and uploads the parts
 concurrently.
func (basics BucketBasics) UploadLargeObject(ctx context.Context, bucketName string,
 objectKey string, largeObject []byte) error {
 largeBuffer := bytes.NewReader(largeObject)
 var partMiBs int64 = 10
 uploader := manager.NewUploader(basics.S3Client, func(u *manager.Uploader) {
  u.PartSize = partMiBs * 1024 * 1024
 })
 _, err := uploader.Upload(ctx, &s3.PutObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
  Body:   largeBuffer,
 })
 if err != nil {
  var apiErr smithy.APIError
  if errors.As(err, &apiErr) && apiErr.ErrorCode() == "EntityTooLarge" {
   log.Printf("Error while uploading object to %s. The object is too large.\n"+
    "The maximum size for a multipart upload is 5TB.", bucketName)
  } else {
   log.Printf("Couldn't upload large object to %v:%v. Here's why: %v\n",
    bucketName, objectKey, err)
  }
 } else {
  err = s3.NewObjectExistsWaiter(basics.S3Client).Wait(
   ctx, &s3.HeadObjectInput{Bucket: aws.String(bucketName), Key:
 aws.String(objectKey)}, time.Minute)
  if err != nil {
   log.Printf("Failed attempt to wait for object %s to exist.\n", objectKey)
  }
 }

 return err
}
```

```
// DownloadLargeObject uses a download manager to download an object from a bucket.
// The download manager gets the data in parts and writes them to a buffer until all
 of
// the data has been downloaded.
func (basics BucketBasics) DownloadLargeObject(ctx context.Context, bucketName
 string, objectKey string) ([]byte, error) {
 var partMiBs int64 = 10
 downloader := manager.NewDownloader(basics.S3Client, func(d *manager.Downloader) {
  d.PartSize = partMiBs * 1024 * 1024
 })
 buffer := manager.NewWriteAtBuffer([]byte{})
 _, err := downloader.Download(ctx, buffer, &s3.GetObjectInput{
  Bucket: aws.String(bucketName),
  Key:    aws.String(objectKey),
 })
 if err != nil {
  log.Printf("Couldn't download large object from %v:%v. Here's why: %v\n",
   bucketName, objectKey, err)
 }
 return buffer.Bytes(), err
}
```

Run an interactive scenario that shows you how to use the upload and download managers in context.

```
import (
 "context"
 "crypto/rand"
 "log"
 "strings"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/s3"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/s3/actions"
)

// RunLargeObjectScenario is an interactive example that shows you how to use Amazon
// Simple Storage Service (Amazon S3) to upload and download large objects.
```

```
//
// 1. Create a bucket.
// 3. Upload a large object to the bucket by using an upload manager.
// 5. Download a large object by using a download manager.
// 8. Delete all objects in the bucket.
// 9. Delete the bucket.
//
// This example creates an Amazon S3 service client from the specified sdkConfig so
 that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunLargeObjectScenario(ctx context.Context, sdkConfig aws.Config, questioner
 demotools.IQuestioner) {
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.")
   _, isMock := questioner.(*demotools.MockQuestioner)
   if isMock || questioner.AskBool("Do you want to see the full error message (y/
n)?", "y") {
    log.Println(r)
   }
  }
 }()

 log.Println(strings.Repeat("-", 88))
 log.Println("Welcome to the Amazon S3 large object demo.")
 log.Println(strings.Repeat("-", 88))

 s3Client := s3.NewFromConfig(sdkConfig)
 bucketBasics := actions.BucketBasics{S3Client: s3Client}

 bucketName := questioner.Ask("Let's create a bucket. Enter a name for your
 bucket:",
  demotools.NotEmpty{})
 bucketExists, err := bucketBasics.BucketExists(ctx, bucketName)
 if err != nil {
  panic(err)
 }
 if !bucketExists {
  err = bucketBasics.CreateBucket(ctx, bucketName, sdkConfig.Region)
  if err != nil {
```

```
  panic(err)
 } else {
  log.Println("Bucket created.")
 }
}
log.Println(strings.Repeat("-", 88))

mibs := 30
log.Printf("Let's create a slice of %v MiB of random bytes and upload it to your
bucket. ", mibs)
questioner.Ask("Press Enter when you're ready.")
largeBytes := make([]byte, 1024*1024*mibs)
_, _ = rand.Read(largeBytes)
largeKey := "doc-example-large"
log.Println("Uploading...")
err = bucketBasics.UploadLargeObject(ctx, bucketName, largeKey, largeBytes)
if err != nil {
 panic(err)
}
log.Printf("Uploaded %v MiB object as %v", mibs, largeKey)
log.Println(strings.Repeat("-", 88))

log.Printf("Let's download the %v MiB object.", mibs)
questioner.Ask("Press Enter when you're ready.")
log.Println("Downloading...")
largeDownload, err := bucketBasics.DownloadLargeObject(ctx, bucketName, largeKey)
if err != nil {
 panic(err)
}
log.Printf("Downloaded %v bytes.", len(largeDownload))
log.Println(strings.Repeat("-", 88))

if questioner.AskBool("Do you want to delete your bucket and all of its "+
 "contents? (y/n)", "y") {
 log.Println("Deleting object.")
 err = bucketBasics.DeleteObjects(ctx, bucketName, []string{largeKey})
 if err != nil {
  panic(err)
 }
 log.Println("Deleting bucket.")
 err = bucketBasics.DeleteBucket(ctx, bucketName)
 if err != nil {
  panic(err)
 }
```

```
 } else {
  log.Println("Okay. Don't forget to delete objects from your bucket to avoid
 charges.")
 }
 log.Println(strings.Repeat("-", 88))

 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Serverless examples

Invoke a Lambda function from an Amazon S3 trigger

The following code example shows how to implement a Lambda function that receives an event triggered by uploading an object to an S3 bucket. The function retrieves the S3 bucket name and object key from the event parameter and calls the Amazon S3 API to retrieve and log the content type of the object.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an S3 event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "log"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
```

```
 "github.com/aws/aws-sdk-go-v2/config"
 "github.com/aws/aws-sdk-go-v2/service/s3"
)

func handler(ctx context.Context, s3Event events.S3Event) error {
 sdkConfig, err := config.LoadDefaultConfig(ctx)
 if err != nil {
  log.Printf("failed to load default config: %s", err)
  return err
 }
 s3Client := s3.NewFromConfig(sdkConfig)

 for _, record := range s3Event.Records {
  bucket := record.S3.Bucket.Name
  key := record.S3.Object.URLDecodedKey
  headOutput, err := s3Client.HeadObject(ctx, &s3.HeadObjectInput{
   Bucket: &bucket,
   Key:    &key,
  })
  if err != nil {
   log.Printf("error getting head of object %s/%s: %s", bucket, key, err)
   return err
  }
  log.Printf("successfully retrieved %s/%s of type %s", bucket, key,
 *headOutput.ContentType)
 }

 return nil
}

func main() {
 lambda.Start(handler)
}
```

Amazon SNS examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon SNS.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Actions
- Scenarios
- Serverless examples

Get started

Hello Amazon SNS

The following code example shows how to get started using Amazon SNS.

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/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/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
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- For API details, see ListTopics in *AWS SDK for Go API Reference*.

Actions

`CreateTopic`

The following code example shows how to use `CreateTopic`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
```

```
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}
```

- For API details, see CreateTopic in *AWS SDK for Go API Reference*.

`DeleteTopic`

The following code example shows how to use `DeleteTopic`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}
```

- For API details, see DeleteTopic in *AWS SDK for Go API Reference*.

`ListTopics`

The following code example shows how to use `ListTopics`.

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/config"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/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
 }
 snsClient := sns.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the topics for your account.")
 var topics []types.Topic
 paginator := sns.NewListTopicsPaginator(snsClient, &sns.ListTopicsInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
  if err != nil {
   log.Printf("Couldn't get topics. Here's why: %v\n", err)
   break
  } else {
   topics = append(topics, output.Topics...)
  }
 }
 if len(topics) == 0 {
  fmt.Println("You don't have any topics!")
 } else {
  for _, topic := range topics {
   fmt.Printf("\t%v\n", *topic.TopicArn)
  }
 }
}
```

- For API details, see ListTopics in *AWS SDK for Go API Reference*.

`Publish`

The following code example shows how to use `Publish`.

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"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
```

```
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
 return err
}
```

- For API details, see Publish in *AWS SDK for Go API Reference*.

`Subscribe`

The following code example shows how to use `Subscribe`.

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.

Subscribe a queue to a topic with optional filters.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
```

```
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

- For API details, see Subscribe in *AWS SDK for Go API Reference*.

Scenarios

Publish messages to queues

The following code example shows how to:

- Create topic (FIFO or non-FIFO).
- Subscribe several queues to the topic with an option to apply a filter.
- Publish messages to the topic.
- Poll the queues for messages received.

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"
 "encoding/json"
```

```
 "fmt"
 "log"
 "strings"
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}

// ScenarioRunner separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string, bool,
 bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or standard.
\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
```

```
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.\n"
 +
   "Deduplication IDs are either set in the message or are automatically generated
\n" +
   "from content using a hash function. If a message is successfully published to\n"
 +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted\n"
 +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to\n"+
   "the topic name.", FIFO_SUFFIX)
 }

 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN) \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS queue.
 ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
```

```
  }
 }
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string, topicArn
 string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)

 err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
 if err != nil {
  panic(err)
 }
 log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
  "messages to it.")
 log.Println(strings.Repeat("-", 88))

 var filterPolicy map[string][]string
 if isFifoTopic {
  if ordinal == "first" {
   log.Println("Subscriptions to a FIFO topic can have filters.\n" +
    "If you add a filter to this subscription, then only the filtered messages\n" +
    "will be received in the queue.\n" +
    "For information about message filtering, see\n" +
    "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
    "For this example, you can filter messages by a \"tone\" attribute.")
  }

  wantFiltering := runner.questioner.AskBool(
   fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
```

```
    "from the %v topic? (y/n) ", queueName, topicName), "y")
  if wantFiltering {
   log.Println("You can filter messages by one or more of the following \"tone\"
 attributes.")

   var toneSelections []string
   askAboutTones := true
   for askAboutTones {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelections = append(toneSelections, ToneChoices[toneIndex])
    askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
 the filter? (y/n) ", "y")
   }
   log.Printf("Your subscription will be filtered to only pass the following tones:
 %v\n", toneSelections)
   filterPolicy = map[string][]string{TONE_KEY: toneSelections}
  }
 }

 subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
 filterPolicy)
 if err != nil {
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn string,
 isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
```

```
  if isFifoTopic {
   log.Println("Because you are using a FIFO topic, you must set a message group ID.
\n" +
    "All messages within the same group will be received in the order they were
 published.")
   groupId = runner.questioner.Ask("Enter a message group ID: ")
   if !contentBasedDeduplication {
    log.Println("Because you are not using content-based deduplication,\n" +
     "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
   }
  }
  if usingFilters {
   if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelection = ToneChoices[toneIndex]
   }
  }

  err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId, TONE_KEY,
 toneSelection)
  if err != nil {
   panic(err)
  }
  log.Println(("Your message was published."))

  publishMore = runner.questioner.AskBool("Do you want to publish another messsage?
 (y/n) ", "y")
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
```

```
    break
   }
   messages = append(messages, currentMsgs...)
  }
  if len(messages) == 0 {
   log.Printf("No messages were received by queue %v.\n", queueUrl)
  } else if len(messages) == 1 {
   log.Printf("One message was received by queue %v:\n", queueUrl)

  } else {
   log.Printf("%v messages were received by queue %v:\n", len(messages), queueUrl)
  }
  for msgIndex, message := range messages {
   messageBody := MessageBody{}
   err := json.Unmarshal([]byte(*message.Body), &messageBody)
   if err != nil {
    panic(err)
   }
   log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
  }

  if len(messages) > 0 {
   log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
   err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
   if err != nil {
    panic(err)
   }
  }
 }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
```

```
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to the
\n"+
  "topic. You can select from several options for configuring the topic and the\n"+
  "subscriptions for the queues. You can then post to the topic and see the results
\n"+
  "in the queues.\n", queueCount)

 log.Println(strings.Repeat("-", 88))

 runner := ScenarioRunner{
  questioner: questioner,
  snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
  sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
 }
 resources.snsActor = runner.snsActor
 resources.sqsActor = runner.sqsActor

 topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
 runner.CreateTopic(ctx)
 resources.topicArn = topicArn
 log.Println(strings.Repeat("-", 88))

 log.Printf("Now you will create %v SQS queues and subscribe them to the topic.\n",
 queueCount)
 ordinals := []string{"first", "next"}
 usingFilters := false
 for _, ordinal := range ordinals {
```

```
  queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
  resources.queueUrls = append(resources.queueUrls, queueUrl)

  _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl, topicName,
 topicArn, ordinal, isFifoTopic)
  usingFilters = usingFilters || filtering
 }

 log.Println(strings.Repeat("-", 88))
 runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
 usingFilters)
 log.Println(strings.Repeat("-", 88))
 runner.PollForMessages(ctx, resources.queueUrls)

 log.Println(strings.Repeat("-", 88))

 wantCleanup := questioner.AskBool("Do you want to remove all AWS resources created
 for this scenario? (y/n) ", "y")
 if wantCleanup {
  log.Println("Cleaning up resources...")
  resources.Cleanup(ctx)
 }

 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps Amazon SNS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)
```

```
// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
```

```
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}
```

```
// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
 return err
}
```

Define a struct that wraps Amazon SQS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"
```

```
 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
 var queueArn string
```

```
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
```

```
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
 }
```

```
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

Clean up resources.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management Console
\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
  if err != nil {
   panic(err)
  }
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Serverless examples

Invoke a Lambda function from an Amazon SNS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SNS topic. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SNS event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main
```

```
import (
 "context"
 "fmt"

 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, snsEvent events.SNSEvent) {
 for _, record := range snsEvent.Records {
  processMessage(record)
 }
 fmt.Println("done")
}

func processMessage(record events.SNSEventRecord) {
 message := record.SNS.Message
 fmt.Printf("Processed message: %s\n", message)
 // TODO: Process your record here
}

func main() {
 lambda.Start(handler)
}
```

Amazon SQS examples using SDK for Go V2

The following code examples show you how to perform actions and implement common scenarios by using the AWS SDK for Go V2 with Amazon SQS.

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.

Each example includes a link to the complete source code, where you can find instructions on how to set up and run the code in context.

Topics

- Get started
- Actions
- Scenarios
- Serverless examples

Get started

Hello Amazon SQS

The following code example shows how to get started using Amazon SQS.

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/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues 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
}
sqsClient := sqs.NewFromConfig(sdkConfig)
fmt.Println("Let's list the queues for your account.")
var queueUrls []string
paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
for paginator.HasMorePages() {
 output, err := paginator.NextPage(ctx)
 if err != nil {
  log.Printf("Couldn't get queues. Here's why: %v\n", err)
  break
 } else {
  queueUrls = append(queueUrls, output.QueueUrls...)
 }
}
if len(queueUrls) == 0 {
 fmt.Println("You don't have any queues!")
} else {
 for _, queueUrl := range queueUrls {
  fmt.Printf("\t%v\n", queueUrl)
 }
}
}
```

- For API details, see ListQueues in *AWS SDK for Go API Reference*.

Actions

`CreateQueue`

The following code example shows how to use `CreateQueue`.

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"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
```

```
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}
```

- For API details, see CreateQueue in *AWS SDK for Go API Reference*.

`DeleteMessageBatch`

The following code example shows how to use `DeleteMessageBatch`.

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"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}
```

```
// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
}
```

- For API details, see DeleteMessageBatch in *AWS SDK for Go API Reference*.

`DeleteQueue`

The following code example shows how to use `DeleteQueue`.

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"
```

```
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

- For API details, see DeleteQueue in *AWS SDK for Go API Reference*.

`GetQueueAttributes`

The following code example shows how to use `GetQueueAttributes`.

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"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}
```

- For API details, see GetQueueAttributes in *AWS SDK for Go API Reference*.

`ListQueues`

The following code example shows how to use `ListQueues`.

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/config"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
)

// main uses the AWS SDK for Go V2 to create an Amazon Simple Queue Service
// (Amazon SQS) client and list the queues 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
 }
 sqsClient := sqs.NewFromConfig(sdkConfig)
 fmt.Println("Let's list the queues for your account.")
 var queueUrls []string
 paginator := sqs.NewListQueuesPaginator(sqsClient, &sqs.ListQueuesInput{})
 for paginator.HasMorePages() {
  output, err := paginator.NextPage(ctx)
```

```
  if err != nil {
   log.Printf("Couldn't get queues. Here's why: %v\n", err)
   break
  } else {
   queueUrls = append(queueUrls, output.QueueUrls...)
  }
 }
 if len(queueUrls) == 0 {
  fmt.Println("You don't have any queues!")
 } else {
  for _, queueUrl := range queueUrls {
   fmt.Printf("\t%v\n", queueUrl)
  }
 }
}
```

- For API details, see ListQueues in *AWS SDK for Go API Reference*.

`ReceiveMessage`

The following code example shows how to use `ReceiveMessage`.

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"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
```

```
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
 }
 return messages, err
}
```

- For API details, see ReceiveMessage in *AWS SDK for Go API Reference*.

`SetQueueAttributes`

The following code example shows how to use `SetQueueAttributes`.

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"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
```

```
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}

// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string
```

- For API details, see SetQueueAttributes in *AWS SDK for Go API Reference*.

Scenarios

Publish messages to queues

The following code example shows how to:

- Create topic (FIFO or non-FIFO).

- Subscribe several queues to the topic with an option to apply a filter.
- Publish messages to the topic.
- Poll the queues for messages received.

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"
 "encoding/json"
 "fmt"
 "log"
 "strings"
 "topics_and_queues/actions"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
 "github.com/awsdocs/aws-doc-sdk-examples/gov2/demotools"
)

const FIFO_SUFFIX = ".fifo"
const TONE_KEY = "tone"

var ToneChoices = []string{"cheerful", "funny", "serious", "sincere"}

// MessageBody is used to deserialize the body of a message from a JSON string.
type MessageBody struct {
 Message string
}
```

```
// ScenarioRunner separates the steps of this scenario into individual functions so
 that
// they are simpler to read and understand.
type ScenarioRunner struct {
 questioner demotools.IQuestioner
 snsActor   *actions.SnsActions
 sqsActor   *actions.SqsActions
}

func (runner ScenarioRunner) CreateTopic(ctx context.Context) (string, string, bool,
 bool) {
 log.Println("SNS topics can be configured as FIFO (First-In-First-Out) or standard.
\n" +
  "FIFO topics deliver messages in order and support deduplication and message
 filtering.")
 isFifoTopic := runner.questioner.AskBool("\nWould you like to work with FIFO
 topics? (y/n) ", "y")

 contentBasedDeduplication := false
 if isFifoTopic {
  log.Println(strings.Repeat("-", 88))
  log.Println("Because you have chosen a FIFO topic, deduplication is supported.\n"
 +
   "Deduplication IDs are either set in the message or are automatically generated
\n" +
   "from content using a hash function. If a message is successfully published to\n"
 +
   "an SNS FIFO topic, any message published and determined to have the same\n" +
   "deduplication ID, within the five-minute deduplication interval, is accepted\n"
 +
   "but not delivered. For more information about deduplication, see:\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/fifo-message-dedup.html.")
  contentBasedDeduplication = runner.questioner.AskBool(
   "\nDo you want to use content-based deduplication instead of entering a
 deduplication ID? (y/n) ", "y")
 }
 log.Println(strings.Repeat("-", 88))

 topicName := runner.questioner.Ask("Enter a name for your SNS topic. ")
 if isFifoTopic {
  topicName = fmt.Sprintf("%v%v", topicName, FIFO_SUFFIX)
  log.Printf("Because you have selected a FIFO topic, '%v' must be appended to\n"+
   "the topic name.", FIFO_SUFFIX)
 }
```

```
 topicArn, err := runner.snsActor.CreateTopic(ctx, topicName, isFifoTopic,
 contentBasedDeduplication)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new topic with the name '%v' and Amazon Resource Name (ARN) \n"+
  "'%v' has been created.", topicName, topicArn)

 return topicName, topicArn, isFifoTopic, contentBasedDeduplication
}

func (runner ScenarioRunner) CreateQueue(ctx context.Context, ordinal string,
 isFifoTopic bool) (string, string) {
 queueName := runner.questioner.Ask(fmt.Sprintf("Enter a name for the %v SQS queue.
 ", ordinal))
 if isFifoTopic {
  queueName = fmt.Sprintf("%v%v", queueName, FIFO_SUFFIX)
  if ordinal == "first" {
   log.Printf("Because you are creating a FIFO SQS queue, '%v' must "+
    "be appended to the queue name.\n", FIFO_SUFFIX)
  }
 }
 queueUrl, err := runner.sqsActor.CreateQueue(ctx, queueName, isFifoTopic)
 if err != nil {
  panic(err)
 }
 log.Printf("Your new SQS queue with the name '%v' and the queue URL "+
  "'%v' has been created.", queueName, queueUrl)

 return queueName, queueUrl
}

func (runner ScenarioRunner) SubscribeQueueToTopic(
 ctx context.Context, queueName string, queueUrl string, topicName string, topicArn
 string, ordinal string,
 isFifoTopic bool) (string, bool) {

 queueArn, err := runner.sqsActor.GetQueueArn(ctx, queueUrl)
 if err != nil {
  panic(err)
 }
 log.Printf("The ARN of your queue is: %v.\n", queueArn)
```

```
err = runner.sqsActor.AttachSendMessagePolicy(ctx, queueUrl, queueArn, topicArn)
if err != nil {
 panic(err)
}
log.Println("Attached an IAM policy to the queue so the SNS topic can send " +
 "messages to it.")
log.Println(strings.Repeat("-", 88))

var filterPolicy map[string][]string
if isFifoTopic {
 if ordinal == "first" {
  log.Println("Subscriptions to a FIFO topic can have filters.\n" +
   "If you add a filter to this subscription, then only the filtered messages\n" +
   "will be received in the queue.\n" +
   "For information about message filtering, see\n" +
   "\thttps://docs.aws.amazon.com/sns/latest/dg/sns-message-filtering.html\n" +
   "For this example, you can filter messages by a \"tone\" attribute.")
 }

 wantFiltering := runner.questioner.AskBool(
  fmt.Sprintf("Do you want to filter messages that are sent to \"%v\"\n"+
   "from the %v topic? (y/n) ", queueName, topicName), "y")
 if wantFiltering {
  log.Println("You can filter messages by one or more of the following \"tone\"
attributes.")

  var toneSelections []string
  askAboutTones := true
  for askAboutTones {
   toneIndex := runner.questioner.AskChoice(
    "Enter the number of the tone you want to filter by:\n", ToneChoices)
   toneSelections = append(toneSelections, ToneChoices[toneIndex])
   askAboutTones = runner.questioner.AskBool("Do you want to add another tone to
the filter? (y/n) ", "y")
  }
  log.Printf("Your subscription will be filtered to only pass the following tones:
%v\n", toneSelections)
  filterPolicy = map[string][]string{TONE_KEY: toneSelections}
 }
}

subscriptionArn, err := runner.snsActor.SubscribeQueue(ctx, topicArn, queueArn,
filterPolicy)
if err != nil {
```

```
  panic(err)
 }
 log.Printf("The queue %v is now subscribed to the topic %v with the subscription
 ARN %v.\n",
  queueName, topicName, subscriptionArn)

 return subscriptionArn, filterPolicy != nil
}

func (runner ScenarioRunner) PublishMessages(ctx context.Context, topicArn string,
 isFifoTopic bool, contentBasedDeduplication bool, usingFilters bool) {
 var message string
 var groupId string
 var dedupId string
 var toneSelection string
 publishMore := true
 for publishMore {
  groupId = ""
  dedupId = ""
  toneSelection = ""
  message = runner.questioner.Ask("Enter a message to publish: ")
  if isFifoTopic {
   log.Println("Because you are using a FIFO topic, you must set a message group ID.
\n" +
    "All messages within the same group will be received in the order they were
 published.")
   groupId = runner.questioner.Ask("Enter a message group ID: ")
   if !contentBasedDeduplication {
    log.Println("Because you are not using content-based deduplication,\n" +
     "you must enter a deduplication ID.")
    dedupId = runner.questioner.Ask("Enter a deduplication ID: ")
   }
  }
  if usingFilters {
   if runner.questioner.AskBool("Add a tone attribute so this message can be
 filtered? (y/n) ", "y") {
    toneIndex := runner.questioner.AskChoice(
     "Enter the number of the tone you want to filter by:\n", ToneChoices)
    toneSelection = ToneChoices[toneIndex]
   }
  }

  err := runner.snsActor.Publish(ctx, topicArn, message, groupId, dedupId, TONE_KEY,
 toneSelection)
```

```
  if err != nil {
   panic(err)
  }
  log.Println(("Your message was published."))

  publishMore = runner.questioner.AskBool("Do you want to publish another messsage?
 (y/n) ", "y")
 }
}

func (runner ScenarioRunner) PollForMessages(ctx context.Context, queueUrls
 []string) {
 log.Println("Polling queues for messages...")
 for _, queueUrl := range queueUrls {
  var messages []types.Message
  for {
   currentMsgs, err := runner.sqsActor.GetMessages(ctx, queueUrl, 10, 1)
   if err != nil {
    panic(err)
   }
   if len(currentMsgs) == 0 {
    break
   }
   messages = append(messages, currentMsgs...)
  }
  if len(messages) == 0 {
   log.Printf("No messages were received by queue %v.\n", queueUrl)
  } else if len(messages) == 1 {
   log.Printf("One message was received by queue %v:\n", queueUrl)

  } else {
   log.Printf("%v messages were received by queue %v:\n", len(messages), queueUrl)
  }
  for msgIndex, message := range messages {
   messageBody := MessageBody{}
   err := json.Unmarshal([]byte(*message.Body), &messageBody)
   if err != nil {
    panic(err)
   }
   log.Printf("Message %v: %v\n", msgIndex+1, messageBody.Message)
  }

  if len(messages) > 0 {
   log.Printf("Deleting %v messages from queue %v.\n", len(messages), queueUrl)
```

```
   err := runner.sqsActor.DeleteMessages(ctx, queueUrl, messages)
   if err != nil {
    panic(err)
   }
  }
 }
}

// RunTopicsAndQueuesScenario is an interactive example that shows you how to use
 the
// AWS SDK for Go to create and use Amazon SNS topics and Amazon SQS queues.
//
// 1. Create a topic (FIFO or non-FIFO).
// 2. Subscribe several queues to the topic with an option to apply a filter.
// 3. Publish messages to the topic.
// 4. Poll the queues for messages received.
// 5. Delete the topic and the queues.
//
// This example creates service clients from the specified sdkConfig so that
// you can replace it with a mocked or stubbed config for unit testing.
//
// It uses a questioner from the `demotools` package to get input during the
 example.
// This package can be found in the ..\..\demotools folder of this repo.
func RunTopicsAndQueuesScenario(
 ctx context.Context, sdkConfig aws.Config, questioner demotools.IQuestioner) {
 resources := Resources{}
 defer func() {
  if r := recover(); r != nil {
   log.Println("Something went wrong with the demo.\n" +
    "Cleaning up any resources that were created...")
   resources.Cleanup(ctx)
  }
 }()
 queueCount := 2

 log.Println(strings.Repeat("-", 88))
 log.Printf("Welcome to messaging with topics and queues.\n\n"+
  "In this scenario, you will create an SNS topic and subscribe %v SQS queues to the
\n"+
  "topic. You can select from several options for configuring the topic and the\n"+
  "subscriptions for the queues. You can then post to the topic and see the results
\n"+
  "in the queues.\n", queueCount)
```

```
log.Println(strings.Repeat("-", 88))

runner := ScenarioRunner{
 questioner: questioner,
 snsActor:   &actions.SnsActions{SnsClient: sns.NewFromConfig(sdkConfig)},
 sqsActor:   &actions.SqsActions{SqsClient: sqs.NewFromConfig(sdkConfig)},
}
resources.snsActor = runner.snsActor
resources.sqsActor = runner.sqsActor

topicName, topicArn, isFifoTopic, contentBasedDeduplication :=
runner.CreateTopic(ctx)
resources.topicArn = topicArn
log.Println(strings.Repeat("-", 88))

log.Printf("Now you will create %v SQS queues and subscribe them to the topic.\n",
queueCount)
ordinals := []string{"first", "next"}
usingFilters := false
for _, ordinal := range ordinals {
 queueName, queueUrl := runner.CreateQueue(ctx, ordinal, isFifoTopic)
 resources.queueUrls = append(resources.queueUrls, queueUrl)

 _, filtering := runner.SubscribeQueueToTopic(ctx, queueName, queueUrl, topicName,
topicArn, ordinal, isFifoTopic)
 usingFilters = usingFilters || filtering
}

log.Println(strings.Repeat("-", 88))
runner.PublishMessages(ctx, topicArn, isFifoTopic, contentBasedDeduplication,
usingFilters)
log.Println(strings.Repeat("-", 88))
runner.PollForMessages(ctx, resources.queueUrls)

log.Println(strings.Repeat("-", 88))

wantCleanup := questioner.AskBool("Do you want to remove all AWS resources created
for this scenario? (y/n) ", "y")
if wantCleanup {
 log.Println("Cleaning up resources...")
 resources.Cleanup(ctx)
}
```

```
 log.Println(strings.Repeat("-", 88))
 log.Println("Thanks for watching!")
 log.Println(strings.Repeat("-", 88))
}
```

Define a struct that wraps Amazon SNS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sns"
 "github.com/aws/aws-sdk-go-v2/service/sns/types"
)

// SnsActions encapsulates the Amazon Simple Notification Service (Amazon SNS)
 actions
// used in the examples.
type SnsActions struct {
 SnsClient *sns.Client
}



// CreateTopic creates an Amazon SNS topic with the specified name. You can
 optionally
// specify that the topic is created as a FIFO topic and whether it uses content-
based
// deduplication instead of ID-based deduplication.
func (actor SnsActions) CreateTopic(ctx context.Context, topicName string,
 isFifoTopic bool, contentBasedDeduplication bool) (string, error) {
 var topicArn string
 topicAttributes := map[string]string{}
 if isFifoTopic {
  topicAttributes["FifoTopic"] = "true"
 }
 if contentBasedDeduplication {
  topicAttributes["ContentBasedDeduplication"] = "true"
```

```
 }
 topic, err := actor.SnsClient.CreateTopic(ctx, &sns.CreateTopicInput{
  Name:       aws.String(topicName),
  Attributes: topicAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create topic %v. Here's why: %v\n", topicName, err)
 } else {
  topicArn = *topic.TopicArn
 }

 return topicArn, err
}



// DeleteTopic delete an Amazon SNS topic.
func (actor SnsActions) DeleteTopic(ctx context.Context, topicArn string) error {
 _, err := actor.SnsClient.DeleteTopic(ctx, &sns.DeleteTopicInput{
  TopicArn: aws.String(topicArn)})
 if err != nil {
  log.Printf("Couldn't delete topic %v. Here's why: %v\n", topicArn, err)
 }
 return err
}



// SubscribeQueue subscribes an Amazon Simple Queue Service (Amazon SQS) queue to an
// Amazon SNS topic. When filterMap is not nil, it is used to specify a filter
 policy
// so that messages are only sent to the queue when the message has the specified
 attributes.
func (actor SnsActions) SubscribeQueue(ctx context.Context, topicArn string,
 queueArn string, filterMap map[string][]string) (string, error) {
 var subscriptionArn string
 var attributes map[string]string
 if filterMap != nil {
  filterBytes, err := json.Marshal(filterMap)
  if err != nil {
   log.Printf("Couldn't create filter policy, here's why: %v\n", err)
   return "", err
  }
  attributes = map[string]string{"FilterPolicy": string(filterBytes)}
```

```
 }
 output, err := actor.SnsClient.Subscribe(ctx, &sns.SubscribeInput{
  Protocol:              aws.String("sqs"),
  TopicArn:              aws.String(topicArn),
  Attributes:            attributes,
  Endpoint:              aws.String(queueArn),
  ReturnSubscriptionArn: true,
 })
 if err != nil {
  log.Printf("Couldn't susbscribe queue %v to topic %v. Here's why: %v\n",
   queueArn, topicArn, err)
 } else {
  subscriptionArn = *output.SubscriptionArn
 }

 return subscriptionArn, err
}



// Publish publishes a message to an Amazon SNS topic. The message is then sent to
 all
// subscribers. When the topic is a FIFO topic, the message must also contain a
 group ID
// and, when ID-based deduplication is used, a deduplication ID. An optional key-
value
// filter attribute can be specified so that the message can be filtered according
 to
// a filter policy.
func (actor SnsActions) Publish(ctx context.Context, topicArn string, message
 string, groupId string, dedupId string, filterKey string, filterValue string) error
 {
 publishInput := sns.PublishInput{TopicArn: aws.String(topicArn), Message:
 aws.String(message)}
 if groupId != "" {
  publishInput.MessageGroupId = aws.String(groupId)
 }
 if dedupId != "" {
  publishInput.MessageDeduplicationId = aws.String(dedupId)
 }
 if filterKey != "" && filterValue != "" {
  publishInput.MessageAttributes = map[string]types.MessageAttributeValue{
   filterKey: {DataType: aws.String("String"), StringValue:
 aws.String(filterValue)},
```

```
  }
 }
 _, err := actor.SnsClient.Publish(ctx, &publishInput)
 if err != nil {
  log.Printf("Couldn't publish message to topic %v. Here's why: %v", topicArn, err)
 }
 return err
}
```

Define a struct that wraps Amazon SQS actions used in this example.

```
import (
 "context"
 "encoding/json"
 "fmt"
 "log"

 "github.com/aws/aws-sdk-go-v2/aws"
 "github.com/aws/aws-sdk-go-v2/service/sqs"
 "github.com/aws/aws-sdk-go-v2/service/sqs/types"
)

// SqsActions encapsulates the Amazon Simple Queue Service (Amazon SQS) actions
// used in the examples.
type SqsActions struct {
 SqsClient *sqs.Client
}



// CreateQueue creates an Amazon SQS queue with the specified name. You can specify
// whether the queue is created as a FIFO queue.
func (actor SqsActions) CreateQueue(ctx context.Context, queueName string,
 isFifoQueue bool) (string, error) {
 var queueUrl string
 queueAttributes := map[string]string{}
 if isFifoQueue {
  queueAttributes["FifoQueue"] = "true"
 }
 queue, err := actor.SqsClient.CreateQueue(ctx, &sqs.CreateQueueInput{
```

```
  QueueName:  aws.String(queueName),
  Attributes: queueAttributes,
 })
 if err != nil {
  log.Printf("Couldn't create queue %v. Here's why: %v\n", queueName, err)
 } else {
  queueUrl = *queue.QueueUrl
 }

 return queueUrl, err
}



// GetQueueArn uses the GetQueueAttributes action to get the Amazon Resource Name
 (ARN)
// of an Amazon SQS queue.
func (actor SqsActions) GetQueueArn(ctx context.Context, queueUrl string) (string,
 error) {
 var queueArn string
 arnAttributeName := types.QueueAttributeNameQueueArn
 attribute, err := actor.SqsClient.GetQueueAttributes(ctx,
 &sqs.GetQueueAttributesInput{
  QueueUrl:       aws.String(queueUrl),
  AttributeNames: []types.QueueAttributeName{arnAttributeName},
 })
 if err != nil {
  log.Printf("Couldn't get ARN for queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  queueArn = attribute.Attributes[string(arnAttributeName)]
 }
 return queueArn, err
}



// AttachSendMessagePolicy uses the SetQueueAttributes action to attach a policy to
 an
// Amazon SQS queue that allows the specified Amazon SNS topic to send messages to
 the
// queue.
func (actor SqsActions) AttachSendMessagePolicy(ctx context.Context, queueUrl
 string, queueArn string, topicArn string) error {
 policyDoc := PolicyDocument{
```

```
  Version: "2012-10-17",
  Statement: []PolicyStatement{{
   Effect:    "Allow",
   Action:    "sqs:SendMessage",
   Principal: map[string]string{"Service": "sns.amazonaws.com"},
   Resource:  aws.String(queueArn),
   Condition: PolicyCondition{"ArnEquals": map[string]string{"aws:SourceArn":
 topicArn}},
  }},
 }
 policyBytes, err := json.Marshal(policyDoc)
 if err != nil {
  log.Printf("Couldn't create policy document. Here's why: %v\n", err)
  return err
 }
 _, err = actor.SqsClient.SetQueueAttributes(ctx, &sqs.SetQueueAttributesInput{
  Attributes: map[string]string{
   string(types.QueueAttributeNamePolicy): string(policyBytes),
  },
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't set send message policy on queue %v. Here's why: %v\n",
 queueUrl, err)
 }
 return err
}

// PolicyDocument defines a policy document as a Go struct that can be serialized
// to JSON.
type PolicyDocument struct {
 Version   string
 Statement []PolicyStatement
}

// PolicyStatement defines a statement in a policy document.
type PolicyStatement struct {
 Effect    string
 Action    string
 Principal map[string]string `json:",omitempty"`
 Resource  *string           `json:",omitempty"`
 Condition PolicyCondition   `json:",omitempty"`
}
```

```
// PolicyCondition defines a condition in a policy.
type PolicyCondition map[string]map[string]string



// GetMessages uses the ReceiveMessage action to get messages from an Amazon SQS
 queue.
func (actor SqsActions) GetMessages(ctx context.Context, queueUrl string,
 maxMessages int32, waitTime int32) ([]types.Message, error) {
 var messages []types.Message
 result, err := actor.SqsClient.ReceiveMessage(ctx, &sqs.ReceiveMessageInput{
  QueueUrl:            aws.String(queueUrl),
  MaxNumberOfMessages: maxMessages,
  WaitTimeSeconds:     waitTime,
 })
 if err != nil {
  log.Printf("Couldn't get messages from queue %v. Here's why: %v\n", queueUrl, err)
 } else {
  messages = result.Messages
 }
 return messages, err
}



// DeleteMessages uses the DeleteMessageBatch action to delete a batch of messages
 from
// an Amazon SQS queue.
func (actor SqsActions) DeleteMessages(ctx context.Context, queueUrl string,
 messages []types.Message) error {
 entries := make([]types.DeleteMessageBatchRequestEntry, len(messages))
 for msgIndex := range messages {
  entries[msgIndex].Id = aws.String(fmt.Sprintf("%v", msgIndex))
  entries[msgIndex].ReceiptHandle = messages[msgIndex].ReceiptHandle
 }
 _, err := actor.SqsClient.DeleteMessageBatch(ctx, &sqs.DeleteMessageBatchInput{
  Entries:  entries,
  QueueUrl: aws.String(queueUrl),
 })
 if err != nil {
  log.Printf("Couldn't delete messages from queue %v. Here's why: %v\n", queueUrl,
 err)
 }
 return err
```

```
}



// DeleteQueue deletes an Amazon SQS queue.
func (actor SqsActions) DeleteQueue(ctx context.Context, queueUrl string) error {
 _, err := actor.SqsClient.DeleteQueue(ctx, &sqs.DeleteQueueInput{
  QueueUrl: aws.String(queueUrl)})
 if err != nil {
  log.Printf("Couldn't delete queue %v. Here's why: %v\n", queueUrl, err)
 }
 return err
}
```

Clean up resources.

```
import (
 "context"
 "fmt"
 "log"
 "topics_and_queues/actions"
)

// Resources keeps track of AWS resources created during an example and handles
// cleanup when the example finishes.
type Resources struct {
 topicArn  string
 queueUrls []string
 snsActor  *actions.SnsActions
 sqsActor  *actions.SqsActions
}

// Cleanup deletes all AWS resources created during an example.
func (resources Resources) Cleanup(ctx context.Context) {
 defer func() {
  if r := recover(); r != nil {
   fmt.Println("Something went wrong during cleanup. Use the AWS Management Console
\n" +
    "to remove any remaining resources that were created for this scenario.")
  }
```

```
 }()

 var err error
 if resources.topicArn != "" {
  log.Printf("Deleting topic %v.\n", resources.topicArn)
  err = resources.snsActor.DeleteTopic(ctx, resources.topicArn)
  if err != nil {
   panic(err)
  }
 }

 for _, queueUrl := range resources.queueUrls {
  log.Printf("Deleting queue %v.\n", queueUrl)
  err = resources.sqsActor.DeleteQueue(ctx, queueUrl)
  if err != nil {
   panic(err)
  }
 }
}
```

- For API details, see the following topics in *AWS SDK for Go API Reference*.
  - CreateQueue
  - CreateTopic
  - DeleteMessageBatch
  - DeleteQueue
  - DeleteTopic
  - GetQueueAttributes
  - Publish
  - ReceiveMessage
  - SetQueueAttributes
  - Subscribe
  - Unsubscribe

Serverless examples

Invoke a Lambda function from an Amazon SQS trigger

The following code example shows how to implement a Lambda function that receives an event triggered by receiving messages from an SQS queue. The function retrieves the messages from the event parameter and logs the content of each message.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Consuming an SQS event with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package integration_sqs_to_lambda

import (
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(event events.SQSEvent) error {
 for _, record := range event.Records {
  err := processMessage(record)
  if err != nil {
   return err
  }
 }
 fmt.Println("done")
 return nil
}

func processMessage(record events.SQSMessage) error {
 fmt.Printf("Processed message %s\n", record.Body)
 // TODO: Do interesting work based on the new message
```

```
 return nil
}

func main() {
 lambda.Start(handler)
}
```

Reporting batch item failures for Lambda functions with an Amazon SQS trigger

The following code example shows how to implement partial batch response for Lambda functions that receive events from an SQS queue. The function reports the batch item failures in the response, signaling to Lambda to retry those messages later.

SDK for Go V2

> **Note**
>
> There's more on GitHub. Find the complete example and learn how to set up and run in the Serverless examples repository.

Reporting SQS batch item failures with Lambda using Go.

```
// Copyright Amazon.com, Inc. or its affiliates. All Rights Reserved.
// SPDX-License-Identifier: Apache-2.0
package main

import (
 "context"
 "fmt"
 "github.com/aws/aws-lambda-go/events"
 "github.com/aws/aws-lambda-go/lambda"
)

func handler(ctx context.Context, sqsEvent events.SQSEvent) (map[string]interface{},
 error) {
 batchItemFailures := []map[string]interface{}{}

 for _, message := range sqsEvent.Records {
  if len(message.Body) > 0 {
```

```
   // Your message processing condition here
   fmt.Printf("Successfully processed message: %s\n", message.Body)
  } else {
   // Message processing failed
   fmt.Printf("Failed to process message %s\n", message.MessageId)
   batchItemFailures = append(batchItemFailures, map[string]interface{}
{"itemIdentifier": message.MessageId})
  }
 }

 sqsBatchResponse := map[string]interface{}{
  "batchItemFailures": batchItemFailures,
 }
 return sqsBatchResponse, nil
}

func main() {
 lambda.Start(handler)
}
```

Migrate to the AWS SDK for Go v2

Minimum Go Version

The AWS SDK for Go requires a minimum version of Go 1.23. The latest version of Go can be downloaded on the Downloads page. See the Release History for more information about each Go version release, and relevant information required for upgrading.

Modularization

The AWS SDK for Go has been updated to take advantage of the Go modules which became the default development mode in Go 1.13. A number of packages provided by the SDK have been modularized and are independently versioned and released respectively. This change enables improved application dependency modeling, and enables the SDK to provide new features and functionality that follows the Go module versioning strategy.

The following list are some Go modules provided by the SDK:

Module	Description
`github.com/aws/aws-sdk-go-v2`	The SDK core
`github.com/aws/aws-sdk-go-v2/config`	Shared Configuration Loading
`github.com/aws/aws-sdk-go-v2/credentials`	AWS Credential Providers
`github.com/aws/aws-sdk-go-v2/feature/ec2/imds`	Amazon EC2 Instance Metadata Service Client

The SDK's service clients and higher level utilities modules are nested under the following import paths:

Import Root	Description
`github.com/aws/aws-sdk-go-v2/service/`	Service Client Modules
`github.com/aws/aws-sdk-go-v2/feature/`	High-Level utilities for services like the Amazon S3 Transfer Manager

Configuration Loading

The session package and associated functionality are replaced with a simplified configuration system provided by the config package. The `config` package is a separate Go module, and can be included in your application's dependencies by with `go get`.

```
go get github.com/aws/aws-sdk-go-v2/config
```

The session.New, session.NewSession, NewSessionWithOptions, and session.Must must be migrated to config.LoadDefaultConfig.

The `config` package provides several helper functions that aid in overriding the shared configuration loading programmatically. These function names are prefixed with `With` followed by option that they override. Let's look at some examples of how to migrate usage of the `session` package.

For more information on loading shared configuration, see Configure the SDK.

Examples

Migrating from NewSession to LoadDefaultConfig

The following example shows how usage of `session.NewSession` without additional argument parameters is migrated to `config.LoadDefaultConfig`.

```
// V1 using NewSession

import "github.com/aws/aws-sdk-go/aws/session"

// ...
```

```
sess, err := session.NewSession()
if err != nil {
    // handle error
}
```

```
// V2 using LoadDefaultConfig

import "context"
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}
```

Migrating from NewSession with aws.Config options

The example shows how to migrate overriding of `aws.Config` values during configuration loading. One or more `config.With*` helper functions can be provided to `config.LoadDefaultConfig` to override the loaded configuration values. In this example the AWS Region is overridden to `us-west-2` using the config.WithRegion helper function.

```
// V1

import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"

// ...

sess, err := session.NewSession(aws.Config{
    Region: aws.String("us-west-2")
})
if err != nil {
    // handle error
}
```

```
// V2
```

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithRegion("us-west-2"),
)
if err != nil {
    // handle error
}
```

Migrating from NewSessionWithOptions

This example shows how to migrate overriding values during configuration loading. Zero or more `config.With*` helper functions can be provided to `config.LoadDefaultConfig` to override the loaded configuration values. In this example we show how to override the target profile that is used when loading the AWS SDK shared configuration.

```
// V1

import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"

// ...

sess, err := session.NewSessionWithOptions(aws.Config{
    Profile: "my-application-profile"
})
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO(),
    config.WithSharedConfigProfile("my-application-profile"),
```

```
)
if err != nil {
    // handle error
}
```

Mocking and `*iface`

The `*iface` packages and interfaces therein (e.g. s3iface.S3API) have been removed. These interface definitions are not stable since they are broken every time a service adds a new operation.

Usage of `*iface` should be replaced by scoped caller-defined interfaces for the service operations being used:

```
// V1

import "io"

import "github.com/aws/aws-sdk-go/service/s3"
import "github.com/aws/aws-sdk-go/service/s3/s3iface"

func GetObjectBytes(client s3iface.S3API, bucket, key string) ([]byte, error) {
    object, err := client.GetObject(&s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return io.ReadAll(object.Body)
}
```

```
// V2

import "context"
import "io"

import "github.com/aws/aws-sdk-go-v2/service/s3"

type GetObjectAPIClient interface {
```

```
    GetObject(context.Context, *s3.GetObjectInput, ...func(*s3.Options))
 (*s3.GetObjectOutput, error)
}

func GetObjectBytes(ctx context.Context, client GetObjectAPIClient, bucket, key string)
 ([]byte, error) {
    object, err := client.GetObject(ctx, &s3.GetObjectInput{
        Bucket: &bucket,
        Key:    &key,
    })
    if err != nil {
        return nil, err
    }
    defer object.Body.Close()

    return io.ReadAll(object.Body)
}
```

See the Unit Testing with the AWS SDK for Go v2 for more information.

Credentials and Credential Providers

The aws/credentials package and associated credential providers have been relocated to the credentials package location. The `credentials` package is a Go module that you retrieve by using `go get`.

```
go get github.com/aws/aws-sdk-go-v2/credentials
```

The AWS SDK for Go v2 release updates the AWS Credential Providers to provide a consistent interface for retrieving AWS Credentials. Each provider implements the aws.CredentialsProvider interface, which defines a `Retrieve` method that returns a `(aws.Credentials, error)`. aws.Credentials that is analogous to the AWS SDK for Go v1 credentials.Value type.

You must wrap `aws.CredentialsProvider` objects with aws.CredentialsCache to allow credential caching to occur. You use NewCredentialsCache to construct a `aws.CredentialsCache` object. By default, credentials configured by `config.LoadDefaultConfig` are wrapped with `aws.CredentialsCache`.

The following table list the location changes of the AWS credential providers from AWS SDK for Go v1 to v2.

Name	V1 Import	V2 Import
Amazon EC2 IAM Role Credentials	`github.com/aws/aws-sdk-go/aws/credentials/ec2rolecreds`	`github.com/aws/aws-sdk-go-v2/credentials/ec2rolecreds`
Endpoint Credentials	`github.com/aws/aws-sdk-go/aws/credentials/endpointcreds`	`github.com/aws/aws-sdk-go-v2/credentials/endpointcreds`
Process Credentials	`github.com/aws/aws-sdk-go/aws/credentials/processcreds`	`github.com/aws/aws-sdk-go-v2/credentials/processcreds`
AWS Security Token Service	`github.com/aws/aws-sdk-go/aws/credentials/stscreds`	`github.com/aws/aws-sdk-go-v2/credentials/stscreds`

Static Credentials

Applications that use credentials.NewStaticCredentials to construct static credential programmatically must use credentials.NewStaticCredentialsProvider.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials"

// ...

appCreds := credentials.NewStaticCredentials(accessKey, secretKey, sessionToken)
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2
```

```
import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials"

// ...

appCreds := aws.NewCredentialsCache(credentials.NewStaticCredentialsProvider(accessKey,
 secretKey, sessionToken))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Amazon EC2 IAM Role Credentials

You must migrate usage of NewCredentials and NewCredentialsWithClient to use New.

The `ec2rolecreds` package's `ec2rolecreds.New` takes functional options of ec2rolecreds.Options as input, allowing you override the specific Amazon EC2 Instance Metadata Service client to use, or to override the credential expiry window.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/ec2rolecreds"

// ...

appCreds := ec2rolecreds.NewCredentials(sess)
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/ec2rolecreds"

// ...
```

```
// New returns an object of a type that satisfies the aws.CredentialProvider interface
appCreds := aws.NewCredentialsCache(ec2rolecreds.New())
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Endpoint Credentials

You must migrate usage of NewCredentialsClient and NewProviderClient to use New.

The `endpointcreds` package's `New` function takes a string argument containing the URL of an HTTP or HTTPS endpoint to retrieve credentials from, and functional options of endpointcreds.Options to mutate the credentials provider and override specific configuration settings.

Process Credentials

You must migrate usage of NewCredentials, NewCredentialsCommand, and NewCredentialsTimeout to use NewProvider or NewProviderCommand.

The `processcreds` package's `NewProvider` function takes a string argument that is the command to be executed in the host environment's shell, and functional options of Options to mutate the credentials provider and override specific configuration settings.

`NewProviderCommand` takes an implementation of the NewCommandBuilder interface that defines more complex process commands that might take one or more command-line arguments, or have certain execution environment requirements. DefaultNewCommandBuilder implements this interface, and defines a command builder for a process that requires multiple command-line arguments.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/processcreds"

// ...

appCreds := processcreds.NewCredentials("/path/to/command")
```

```
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/processcreds"

// ...

appCreds := aws.NewCredentialsCache(processcreds.NewProvider("/path/to/command"))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

AWS Security Token Service Credentials

AssumeRole

You must migrate usage of NewCredentials, and NewCredentialsWithClient to use NewAssumeRoleProvider.

The `stscreds` package's `NewAssumeRoleProvider` function must be called with a sts.Client, and the AWS Identity and Access Management Role ARN to be assumed from the provided `sts.Client`'s configured credentials. You can also provide a set of functional options of AssumeRoleOptions to modify other optional settings of the provider.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/stscreds"

// ...

appCreds := stscreds.NewCredentials(sess, "arn:aws:iam::123456789012:role/demo")
value, err := appCreds.Get()
if err != nil {
```

```
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/credentials/stscreds"
import "github.com/aws/aws-sdk-go-v2/service/sts"

// ...

client := sts.NewFromConfig(cfg)

appCreds := stscreds.NewAssumeRoleProvider(client, "arn:aws:iam::123456789012:role/
demo")
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

AssumeRoleWithWebIdentity

You must migrate usage of NewWebIdentityCredentials, NewWebIdentityRoleProvider, and NewWebIdentityRoleProviderWithToken to use NewWebIdentityRoleProvider.

The `stscreds` package's `NewWebIdentityRoleProvider` function must be called with a sts.Client, and the AWS Identity and Access Management Role ARN to be assumed using the provided `sts.Client`'s configured credentials, and an implementation of a IdentityTokenRetriever for providing the OAuth 2.0 or OpenID Connect ID token. IdentityTokenFile is an `IdentityTokenRetriever` that can be used to provide the web identity token from a file located on the application's host file-system. You can also provide a set of functional options of WebIdentityRoleOptions to modify other optional settings for the provider.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/credentials/stscreds"

// ...
```

```
appCreds := stscreds.NewWebIdentityRoleProvider(sess, "arn:aws:iam::123456789012:role/
demo", "sessionName", "/path/to/token")
value, err := appCreds.Get()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/aws"
import "github.com/aws/aws-sdk-go-v2/credentials/stscreds"
import "github.com/aws/aws-sdk-go-v2/service/sts"

// ...

client := sts.NewFromConfig(cfg)

appCreds := aws.NewCredentialsCache(stscreds.NewWebIdentityRoleProvider(
        client,
        "arn:aws:iam::123456789012:role/demo",
        stscreds.IdentityTokenFile("/path/to/file"),
        func(o *stscreds.WebIdentityRoleOptions) {
            o.RoleSessionName = "sessionName"
        }))
value, err := appCreds.Retrieve(context.TODO())
if err != nil {
    // handle error
}
```

Service Clients

AWS SDK for Go provides service client modules nested under the `github.com/aws/aws-sdk-go-v2/service` import path. Each service client is contained in a Go package using each service's unique identifier. The following table provides some examples of service import paths in the AWS SDK for Go.

Service Name	V1 Import Path	V2 Import Path
Amazon S3	`github.com/aws/aws-sdk-go/service/s3`	`github.com/aws/aws-sdk-go-v2/service/s3`
Amazon DynamoDB	`github.com/aws/aws-sdk-go/service/dynamodb`	`github.com/aws/aws-sdk-go-v2/service/dynamodb`
Amazon CloudWatch Logs	`github.com/aws/aws-sdk-go/service/cloudwatchlogs`	`github.com/aws/aws-sdk-go-v2/service/cloudwatchlogs`

Each service client package is an independently versioned Go module. To add the service client as a dependency of your application, use the `go get` command with the service's import path. For example, to add the Amazon S3 client to your dependencies use

```
go get github.com/aws/aws-sdk-go-v2/service/s3
```

Client Construction

You can construct clients in the AWS SDK for Go using either the `New` or `NewFromConfig` constructor functions in the client's package. When migrating from the AWS SDK for Go v1, we recommend that you use the `NewFromConfig` variant, which will return a new service client using values from an `aws.Config`. The `aws.Config` value will have been created while loading the SDK shared configuration using `config.LoadDefaultConfig`. For details on creating service clients, see Use the AWS SDK for Go v2 with AWS services.

Example 1

```
// V1

import "github.com/aws/aws-sdk-go/aws/session"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

sess, err := session.NewSession()
```

```
if err != nil {
    // handle error
}

client := s3.New(sess)
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg)
```

Example 2: Overriding Client Settings

```
// V1

import "github.com/aws/aws-sdk-go/aws"
import "github.com/aws/aws-sdk-go/aws/session"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

sess, err := session.NewSession()
if err != nil {
    // handle error
}

client := s3.New(sess, &aws.Config{
    Region: aws.String("us-west-2"),
})
```

```
// V2
```

```
import "context"
import "github.com/aws/aws-sdk-go-v2/config"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

cfg, err := config.LoadDefaultConfig(context.TODO())
if err != nil {
    // handle error
}

client := s3.NewFromConfig(cfg, func(o *s3.Options) {
    o.Region = "us-west-2"
})
```

Endpoints

The endpoints package no longer exists in the AWS SDK for Go. Each service client now embeds its required AWS endpoint metadata within the client package. This reduces the overall binary size of compiled applications by no longer including endpoint metadata for services not used by your application.

Additionally, each service now exposes its own interface for endpoint resolution in `EndpointResolverV2`. Each API takes a unique set of parameters for a service `EndpointParameters`, the values of which are sourced by the SDK from various locations when an operation is invoked.

By default, service clients use their configured AWS Region to resolve the service endpoint for the target Region. If your application requires a custom endpoint, you can specify custom behavior on `EndpointResolverV2` field on the `aws.Config` structure. If your application implements a custom endpoints.Resolver you must migrate it to conform to this new per-service interface.

For more information on endpoints and implementing a custom resolver, see Configure Client Endpoints.

Authentication

The AWS SDK for Go supports more advanced authentication behavior, which enables the use of newer AWS service features such as codecatalyst and S3 Express One Zone. Additionally, this behavior can be customized on a per-client basis.

Invoking API Operations

The number of service client operation methods have been reduced significantly. The `<OperationName>Request`, `<OperationName>WithContext`, and `<OperationName>` methods have all been consolidated into single operation method, `<OperationName>`.

Example

The following example shows how calls to the Amazon S3 PutObject operation would be migrated from AWS SDK for Go v1 to v2.

```
// V1

import "context"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

// Pattern 1
output, err := client.PutObject(&s3.PutObjectInput{
    // input parameters
})

// Pattern 2
output, err := client.PutObjectWithContext(context.TODO(), &s3.PutObjectInput{
    // input parameters
})

// Pattern 3
req, output := client.PutObjectRequest(context.TODO(), &s3.PutObjectInput{
    // input parameters
})
err := req.Send()
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...
```

```
client := s3.NewFromConfig(cfg)

output, err := client.PutObject(context.TODO(), &s3.PutObjectInput{
    // input parameters
})
```

Service Data Types

The top-level input and output types of an operation are found in the service client package. The input and output type for a given operation follow the pattern of `<OperationName>Input` and `<OperationName>Output`, where `OperationName` is the name of the operation you are invoking. For example, the input and output shape for the Amazon S3 PutObject operation are PutObjectInput and PutObjectOutput respectively.

All other service data types, other than input and output types, have been migrated to the `types` package located under the service client package import path hierarchy. For example, the s3.AccessControlPolicy type is now located at types.AccessControlPolicy.

Enumeration Values

The SDK now provides a typed experience for all API enumeration fields. Rather than using a string literal value copied from the service API reference documentation, you can now use one of the concrete types found in the service client's `types` package. For example, you can provide the Amazon S3 PutObjectInput operation with an ACL to be applied on an object. In the AWS SDK for Go v1, this parameter was a `*string` type. In the AWS SDK for Go, this parameter is now a types.ObjectCannedACL. The `types` package provides generated constants for the valid enumeration values that can be assigned to this field. For example types.ObjectCannedACLPrivate is the constant for the "private" canned ACL value. This value can be used in place of managing string constants within your application.

Pointer Parameters

The AWS SDK for Go v1 required pointer references to be passed for all input parameters to service operations. The AWS SDK for Go v2 has simplified the experience with most services by removing the need to pass input values as pointers where possible. This change means that many service clients operations no longer require your application to pass pointer references for the following types: `uint8`, `uint16`, `uint32`, `int8`, `int16`, `int32`, `float32`, `float64`, `bool`. Similarly, slice

and map element types have been updated accordingly to reflect whether their elements must be passed as pointer references.

The aws package contains helper functions for creating pointers for the Go built-in types, these helpers should be used to more easily handle creating pointer types for these Go types. Similarly, helper methods are provided for safely de-referencing pointer values for these types. For example, the aws.String function converts from `string` ⇒ `*string`. Inversely, the aws.ToString converts from `*string` ⇒ `string`. When upgrading your application from AWS SDK for Go v1 to v2, you must migrate usage of the helpers for converting from the pointer types to the non-pointer variants. For example, aws.StringValue must be updated to `aws.ToString`.

Errors Types

The AWS SDK for Go takes full advantage of the error wrapping functionality introduced in Go 1.13. Services that model error responses have generated types available in their client's `types` package that can be used to test whether a client operation error was caused by one of these types. For example, Amazon S3 `GetObject` operation can return a `NoSuchKey` error if attempting to retrieve an object key that doesn't exist. You can use errors.As to test whether the returned operation error is a types.NoSuchKey error. In the event a service does not model a specific type for an error, you can utilize the smithy.APIError interface type for inspecting the returned error code and message from the service. This functionality replaces awserr.Error and the other awserr functionality from the AWS SDK for Go v1. For more details information on handling errors, see Handling Errors in the AWS SDK for Go V2.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/awserr"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

output, err := s3.GetObject(&s3.GetObjectInput{
    // input parameters
})
if err != nil {
    if awsErr, ok := err.(awserr.Error); ok {
```

```
        if awsErr.Code() == "NoSuchKey" {
            // handle NoSuchKey
        } else {
            // handle other codes
        }
        return
    }
    // handle a error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/service/s3"
import "github.com/aws/aws-sdk-go-v2/service/s3/types"
import "github.com/aws/smithy-go"

// ...

client := s3.NewFromConfig(cfg)

output, err := client.GetObject(context.TODO(), &s3.GetObjectInput{
    // input parameters
})
if err != nil {
    var nsk *types.NoSuchKey
    if errors.As(err, &nsk) {
        // handle NoSuchKey error
        return
    }
    var apiErr smithy.APIError
    if errors.As(err, &apiErr) {
        code := apiErr.ErrorCode()
        message := apiErr.ErrorMessage()
        // handle error code
        return
    }
    // handle error
    return
}
```

Paginators

Service operation paginators are no longer invoked as methods on the service client. To use a paginator for an operation you must construct a paginator for an operation using one of the paginator constructor methods. For example, to use paginate over the Amazon S3 `ListObjectsV2` operation you must construct its paginator using the s3.NewListObjectsV2Paginator. This constructor returns a ListObjectsV2Paginator which provides the methods `HasMorePages`, and `NextPage` for determining whether there are more pages to retrieve and invoking the operation to retrieve the next page respectively. More details on using the SDK paginators can be found at Using Operation Paginators.

Let's look at an example of how to migrate from a AWS SDK for Go v1 paginator to the AWS SDK for Go v2 equivalent.

Example

```
// V1

import "fmt"
import "github.com/aws/aws-sdk-go/service/s3"

// ...

client := s3.New(sess)

params := &s3.ListObjectsV2Input{
    // input parameters
}

totalObjects := 0
err := client.ListObjectsV2Pages(params, func(output *s3.ListObjectsV2Output, lastPage
 bool) bool {
    totalObjects += len(output.Contents)
    return !lastPage
})
if err != nil {
    // handle error
}
fmt.Println("total objects:", totalObjects)
```

```
// V2
```

```
import "context"
import "fmt"
import "github.com/aws/aws-sdk-go-v2/service/s3"

// ...

client := s3.NewFromConfig(cfg)

params := &s3.ListObjectsV2Input{
    // input parameters
}

totalObjects := 0
paginator := s3.NewListObjectsV2Paginator(client, params)
for paginator.HasMorePages() {
    output, err := paginator.NextPage(context.TODO())
    if err != nil {
        // handle error
    }
    totalObjects += len(output.Contents)
}
fmt.Println("total objects:", totalObjects)
```

Waiters

Service operation waiters are no longer invoked as methods on the service client. To use a waiter you first construct the desired waiter type, and then invoke the wait method. For example, to wait for a Amazon S3 Bucket to exist, you must construct a `BucketExists` waiter. Use the s3.NewBucketExistsWaiter constructor to create a s3.BucketExistsWaiter. The `s3.BucketExistsWaiter` provides a `Wait` method which can be used to wait for a bucket to become available.

Presigned Requests

The V1 SDK technically supported presigning *any* AWS SDK operation, however, this does not accurately represent what is actually supported at the service level (and in reality most AWS service operations do not support presigning).

AWS SDK for Go resolves this by exposing specific `PresignClient` implementations in service packages with specific APIs for supported presignable operations.

Note: If a service is missing presigning support for an operation that you were successfully using in SDK v1, please let us know by filing an issue on GitHub.

Uses of Presign and PresignRequest must be converted to use service-specific presigning clients.

The following example shows how to migrate presigning of an S3 GetObject request:

```
// V1

import (
    "fmt"
    "time"

    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func main() {
    sess := session.Must(session.NewSessionWithOptions(session.Options{
        SharedConfigState: session.SharedConfigEnable,
    }))

    svc := s3.New(sess)
    req, _ := svc.GetObjectRequest(&s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("key"),
    })

    // pattern 1
    url1, err := req.Presign(20 * time.Minute)
    if err != nil {
        panic(err)
    }
    fmt.Println(url1)

    // pattern 2
    url2, header, err := req.PresignRequest(20 * time.Minute)
    if err != nil {
        panic(err)
    }
    fmt.Println(url2, header)
}
```

```
// V2

import (
    "context"
    "fmt"
    "time"

    "github.com/aws/aws-sdk-go-v2/aws"
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
)

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        panic(err)
    }

    svc := s3.NewPresignClient(s3.NewFromConfig(cfg))
    req, err := svc.PresignGetObject(context.Background(), &s3.GetObjectInput{
        Bucket: aws.String("amzn-s3-demo-bucket"),
        Key:    aws.String("key"),
    }, func(o *s3.PresignOptions) {
        o.Expires = 20 * time.Minute
    })
    if err != nil {
        panic(err)
    }

    fmt.Println(req.Method, req.URL, req.SignedHeader)
}
```

Request customization

The monolithic request.Request API has been re-compartmentalized.

Operation input/output

The opaque `Request` fields `Params` and `Data`, which hold the operation input and output structures respectively, are now accessible within specific middleware phases as input/output:

Request handlers which reference `Request.Params` and `Request.Data` must be migrated to middleware.

migrating `Params`

```
// V1

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func withPutObjectDefaultACL(acl string) request.Option {
    return func(r *request.Request) {
        in, ok := r.Params.(*s3.PutObjectInput)
        if !ok {
            return
        }

        if in.ACL == nil {
            in.ACL = aws.String(acl)
        }
        r.Params = in
    }
}

func main() {
    sess := session.Must(session.NewSession())

 sess.Handlers.Validate.PushBack(withPutObjectDefaultACL(s3.ObjectCannedACLBucketOwnerFullContr

    // ...
}
```

```
// V2

import (
    "context"

    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/aws-sdk-go-v2/service/s3/types"
```

```
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type withPutObjectDefaultACL struct {
    acl types.ObjectCannedACL
}

// implements middleware.InitializeMiddleware, which runs BEFORE a request has
// been serialized and can act on the operation input
var _ middleware.InitializeMiddleware = (*withPutObjectDefaultACL)(nil)

func (*withPutObjectDefaultACL) ID() string {
    return "withPutObjectDefaultACL"
}

func (m *withPutObjectDefaultACL) HandleInitialize(ctx context.Context, in
 middleware.InitializeInput, next middleware.InitializeHandler) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    input, ok := in.Parameters.(*s3.PutObjectInput)
    if !ok {
        return next.HandleInitialize(ctx, in)
    }

    if len(input.ACL) == 0 {
        input.ACL = m.acl
    }
    in.Parameters = input
    return next.HandleInitialize(ctx, in)
}

// create a helper function to simplify instrumentation of our middleware
func WithPutObjectDefaultACL(acl types.ObjectCannedACL) func (*s3.Options) {
    return func(o *s3.Options) {
        o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
            return s.Initialize.Add(&withPutObjectDefaultACL{acl: acl},
 middleware.After)
        })
    }
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
```

```
    if err != nil {
        // ...
    }

    svc := s3.NewFromConfig(cfg,
 WithPutObjectDefaultACL(types.ObjectCannedACLBucketOwnerFullControl))
    // ...
}
```

migrating `Data`

```
// V1

import (
    "github.com/aws/aws-sdk-go/aws"
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
    "github.com/aws/aws-sdk-go/service/s3"
)

func readPutObjectOutput(r *request.Request) {
        output, ok := r.Data.(*s3.PutObjectOutput)
        if !ok {
            return
        }

        // ...
    }
}

func main() {
    sess := session.Must(session.NewSession())
    sess.Handlers.Unmarshal.PushBack(readPutObjectOutput)

    svc := s3.New(sess)
    // ...
}
```

```
// V2

import (
    "context"
```

```
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type readPutObjectOutput struct{}

var _ middleware.DeserializeMiddleware = (*readPutObjectOutput)(nil)

func (*readPutObjectOutput) ID() string {
    return "readPutObjectOutput"
}

func (*readPutObjectOutput) HandleDeserialize(ctx context.Context, in
 middleware.DeserializeInput, next middleware.DeserializeHandler) (
    out middleware.DeserializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleDeserialize(ctx, in)
    if err != nil {
        // ...
    }

    output, ok := in.Parameters.(*s3.PutObjectOutput)
    if !ok {
        return out, metadata, err
    }

    // inspect output...

    return out, metadata, err
}

func WithReadPutObjectOutput(o *s3.Options) {
    o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
        return s.Initialize.Add(&withReadPutObjectOutput{}, middleware.Before)
    })
}

func main() {
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        // ...
    }
```

```
    svc := s3.NewFromConfig(cfg, WithReadPutObjectOutput)
    // ...
}
```

HTTP request/response

The HTTPRequest and HTTPResponse fields from Request are now exposed in specific middleware phases. Since middleware is transport-agnostic, you must perform a type assertion on the middleware input or output to reveal the underlying HTTP request or response.

Request handlers which reference Request.HTTPRequest and Request.HTTPResponse must be migrated to middleware.

migrating HTTPRequest

```
// V1

import (
    "github.com/aws/aws-sdk-go/aws/request"
    "github.com/aws/aws-sdk-go/aws/session"
)

func withHeader(header, val string) request.Option {
    return func(r *request.Request) {
        request.HTTPRequest.Header.Set(header, val)
    }
}

func main() {
    sess := session.Must(session.NewSession())
    sess.Handlers.Build.PushBack(withHeader("x-user-header", "..."))

    svc := s3.New(sess)
    // ...
}
```

```
// V2

import (
    "context"
    "fmt"
```

```
    "github.com/aws/aws-sdk-go-v2/config"
    "github.com/aws/aws-sdk-go-v2/service/s3"
    "github.com/aws/smithy-go/middleware"
    smithyhttp "github.com/aws/smithy-go/transport/http"
)

type withHeader struct {
    header, val string
}

// implements middleware.BuildMiddleware, which runs AFTER a request has been
// serialized and can operate on the transport request
var _ middleware.BuildMiddleware = (*withHeader)(nil)

func (*withHeader) ID() string {
    return "withHeader"
}

func (m *withHeader) HandleBuild(ctx context.Context, in middleware.BuildInput, next
 middleware.BuildHandler) (
    out middleware.BuildOutput, metadata middleware.Metadata, err error,
) {
    req, ok := in.Request.(*smithyhttp.Request)
    if !ok {
        return out, metadata, fmt.Errorf("unrecognized transport type %T", in.Request)
    }

    req.Header.Set(m.header, m.val)
    return next.HandleBuild(ctx, in)
}

func WithHeader(header, val string) func (*s3.Options) {
    return func(o *s3.Options) {
        o.APIOptions = append(o.APIOptions, func (s *middleware.Stack) error {
            return s.Build.Add(&withHeader{
                header: header,
                val: val,
            }, middleware.After)
        })
    }
}

func main() {
```

```
    cfg, err := config.LoadDefaultConfig(context.Background())
    if err != nil {
        // ...
    }

    svc := s3.NewFromConfig(cfg, WithHeader("x-user-header", "..."))
    // ...
}
```

Handler phases

SDK v2 middleware phases are the successor to v1 handler phases.

The following table provides a rough mapping of v1 handler phases to their equivalent location within the V2 middleware stack:

v1 handler name	v2 middleware phase
Validate	Initialize
Build	Serialize
Sign	Finalize
Send	n/a (1)
ValidateResponse	Deserialize
Unmarshal	Deserialize
UnmarshalMetadata	Deserialize
UnmarshalError	Deserialize
Retry	Finalize, after `"Retry"` middleware (2)
AfterRetry	Finalize, before `"Retry"` middleware, post-`next.HandleFinalize()` (2,3)
CompleteAttempt	Finalize, end of step

v1 handler name	v2 middleware phase
Complete	Initialize, start of step, post-`next.HandleInitialize()` (3)

(1) The Send phase in v1 is effectively the wrapped HTTP client round-trip in v2. This behavior is controlled by the `HTTPClient` field on client options.

(2) Any middleware after the `"Retry"` middleware in the Finalize step will be part of the retry loop.

(3) The middleware "stack" at operation time is built into a repeatedly-decorated handler function. Each handler is responsible for calling the next one in the chain. This implicitly means that a middleware step can also take action AFTER its next step has been called.

For example, for the Initialize step, which is at the top of the stack, this means Initialize middlewares that take action after calling the next handler effectively operate at the end of the request:

```
// V2

import (
    "context"

    "github.com/aws/smithy-go/middleware"
)

type onComplete struct{}

var _ middleware.InitializeMiddleware = (*onComplete)(nil)

func (*onComplete) ID() string {
    return "onComplete"
}

func (*onComplete) HandleInitialize(ctx context.Context, in middleware.InitializeInput,
 next middleware.InitializeHandler) (
    out middleware.InitializeOutput, metadata middleware.Metadata, err error,
) {
    out, metadata, err = next.HandleInitialize(ctx, in)
```

```
    // the entire operation was invoked above - the deserialized response is
    // available opaquely in out.Result, run post-op actions here...

    return out, metadata, err
}
```

Features

Amazon EC2 Instance Metadata Service

The AWS SDK for Go provides an Amazon EC2 Instance Metadata Service (IMDS) client that you can use to query the local IMDS when executing your application on an Amazon EC2 instance. The IMDS client is a separate Go module that can be added to your application by using

```
go get github.com/aws/aws-sdk-go-v2/feature/ec2/imds
```

The client constructor and method operations have been updated to match the design of the other SDK service clients.

Example

```
// V1

import "github.com/aws/aws-sdk-go/aws/ec2metadata"

// ...

client := ec2metadata.New(sess)

region, err := client.Region()
if err != nil {
    // handle error
}
```

```
// V2

import "context"
import "github.com/aws/aws-sdk-go-v2/feature/ec2/imds"

// ...
```

```
client := imds.NewFromConfig(cfg)

region, err := client.GetRegion(context.TODO())
if err != nil {
    // handle error
}
```

Amazon S3 Transfer Manager

The Amazon S3 transfer manager is available for managing uploads and downloads of objects concurrently. This package is located in a Go module outside the service client import path. This module can be retrieved by using `go get github.com/aws/aws-sdk-go-v2/feature/s3/manager`.

s3.NewUploader and s3.NewUploaderWithClient have been replaced with the constructor method manager.NewUploader for creating an Upload manager client.

s3.NewDownloader and s3.NewDownloaderWithClient have been replaced with a single constructor method manager.NewDownloader for creating a Download manager client.

Amazon CloudFront Signing Utilities

The AWS SDK for Go provides Amazon CloudFront signing utilities in a Go module outside the service client import path. This module can be retrieved by using `go get`.

```
go get github.com/aws/aws-sdk-go-v2/feature/cloudfront/sign
```

Amazon S3 Encryption Client

Starting in AWS SDK for Go, the Amazon S3 encryption client is a separate module under AWS Crypto Tools. The latest version of the S3 encryption client for Go, 3.x, is now available at GitHub. This module can be retrieved by using `go get`:

```
go get github.com/aws/amazon-s3-encryption-client-go/v3
```

The separate `EncryptionClient` (v1, v2) and `DecryptionClient` (v1, v2) APIs have been replaced with a single client, S3EncryptionClientV3, that exposes both encrypt and decrypt functionality.

Like other service clients in AWS SDK for Go, the operation APIs have been condensed:

- The `GetObject`, `GetObjectRequest`, and `GetObjectWithContext` decryption APIs are replaced by GetObject.
- The `PutObject`, `PutObjectRequest`, and `PutObjectWithContext` encryption APIs are replaced by PutObject.

To learn how to migrate to the 3.x major version of the encryption client, see this guide.

Service Customizations Changes

Amazon S3

When migrating from AWS SDK for Go v1 to v2, an important change to be aware of involves the handling of the `SSECustomerKey` used for server-side encryption with customer-provided keys (SSE-C). In AWS SDK for Go v1, the encoding of the `SSECustomerKey` to Base64 was handled internally by the SDK. In SDK v2, this automatic encoding has been removed, and it is now required to manually encode the `SSECustomerKey` to Base64 before passing it to the SDK.

Example Adjustment:

```
// V1

import (
  "context"
  "encoding/base64"
  "github.com/aws/aws-sdk-go-v2/config"
  "github.com/aws/aws-sdk-go-v2/service/s3"
)
// ... more code

plainTextKey := "12345678901234567890123456789012" // 32 bytes in length

// calculate md5..

_, err = client.PutObjectWithContext(context.Background(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("your-object-key"),
    Body:                 strings.NewReader("hello-world"),
    SSECustomerKey:       &plainTextKey,
```

```
    SSECustomerKeyMD5:    &base64Md5,
    SSECustomerAlgorithm: aws.String("AES256"),
})

// ... more code
```

```
// V2

import (
  "github.com/aws/aws-sdk-go/aws"
  "github.com/aws/aws-sdk-go/aws/session"
  "github.com/aws/aws-sdk-go/service/s3"
)

// ... more code

plainTextKey := "12345678901234567890123456789012" // 32 bytes in length
base64EncodedKey := base64.StdEncoding.EncodeToString([]byte(plainTextKey))

// calculate md5..

_, err = client.PutObject(context.Background(), &s3.PutObjectInput{
    Bucket: aws.String("amzn-s3-demo-bucket"),
    Key:    aws.String("your-object-key"),
    Body:                 strings.NewReader("hello-world"),
    SSECustomerKey:       &base64EncodedKey,
    SSECustomerKeyMD5:    &base64Md5,
    SSECustomerAlgorithm: aws.String("AES256"),
})

// ... more code
```

Security in AWS SDK for Go

Cloud security at AWS is the highest priority. As an AWS customer, you benefit from data centers and network architectures that are 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 AWS SDK for Go. The following topics show you how to configure AWS SDK for Go to meet your security and compliance objectives. You also learn how to use other AWS services that help you to monitor and secure your AWS SDK for Go resources.

Topics

- Data protection in AWS SDK for Go
- Compliance validation for AWS SDK for Go
- Resilience in AWS SDK for Go
- Enforcing a minimum TLS version

Data protection in AWS SDK for Go

The AWS shared responsibility model applies to data protection in AWS SDK for Go. 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. You are also responsible for the security configuration and management tasks for the AWS services that you use. For more information about data privacy, see Data Privacy FAQ. For

information about data protection in Europe, see the General Data Protection Regulation (GDPR) Center.

For data protection purposes, we recommend that you protect AWS account credentials and set up individual users with AWS IAM Identity Center or 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. We require TLS 1.2 and recommend TLS 1.3.
- Set up API and user activity logging with AWS CloudTrail. For information about using CloudTrail trails to capture AWS activities, see Working with CloudTrail trails in the *AWS CloudTrail User Guide*.
- 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 sensitive data that is stored in Amazon S3.
- If you require FIPS 140-3 validated cryptographic modules when accessing AWS through a command line interface or an API, use a FIPS endpoint. For more information about the available FIPS endpoints, see Federal Information Processing Standard (FIPS) 140-3.

We strongly recommend that you never put confidential or sensitive information, such as your customers' email addresses, into tags or free-form text fields such as a **Name** field. This includes when you work with AWS SDK for Go or other AWS services using the console, API, AWS CLI, or AWS SDKs. Any data that you enter into tags or free-form text fields used for names may be used for billing or diagnostic logs. If you provide a URL to an external server, we strongly recommend that you do not include credentials information in the URL to validate your request to that server.

Compliance validation for AWS SDK for Go

To learn whether an AWS service is within the scope of specific compliance programs, see AWS services in Scope by Compliance Program and choose the compliance program that you are interested in. For general information, see AWS Compliance Programs.

You can download third-party audit reports using AWS Artifact. For more information, see Downloading Reports in AWS Artifact.

Your compliance responsibility when using AWS services is determined by the sensitivity of your data, your company's compliance objectives, and applicable laws and regulations. For more information about your compliance responsibility when using AWS services, see AWS Security Documentation.

Resilience in AWS SDK for Go

The AWS global infrastructure is built around AWS Regions and Availability Zones. AWS Regions provide multiple physically separated and isolated Availability Zones, which are connected with 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.

Enforcing a minimum TLS version

To add increased security when communicating with AWS services, you must configure your client to use TLS 1.2 or later. To work with AWS services, the underlying Go runtime must support a minimum version of TLS 1.2, but TLS 1.3 is recommended.

TLS 1.3 is the prerequisite to enable post-quantum cryptography, which might require additional actions or configurations. For more information, see Enabling hybrid post-quantum TLS in the AWS SDKs and Tools Reference Guide.

> **Note**
>
> As of Go 1.18 on the Go website, the TLS configuration used by `net/http` defaults to a minimum of TLS 1.2 and disables support for TLS 1.0 and TLS 1.1.

Setting the TLS version

In the AWS SDK for Go v2, you configure the minimum TLS version by providing a custom HTTP client whose transport specifies a tls.Config, which is documented on the Go package documentation website. For more information about customizing the HTTP client, see Customize the HTTP Client.

The following example builds an HTTP client that requires a minimum of TLS 1.2 and passes it to `config.LoadDefaultConfig`.

```
import (
    "context"
    "crypto/tls"
    "log"
    "net/http"

    awshttp "github.com/aws/aws-sdk-go-v2/aws/transport/http"
    "github.com/aws/aws-sdk-go-v2/config"
)

// ...

httpClient := awshttp.NewBuildableClient().WithTransportOptions(func(tr
 *http.Transport) {
    tr.TLSClientConfig = &tls.Config{
        MinVersion: tls.VersionTLS12,
    }
})

cfg, err := config.LoadDefaultConfig(context.TODO(), config.WithHTTPClient(httpClient))
if err != nil {
    log.Fatalf("failed to load configuration, %v", err)
}
```

To require TLS 1.3, set `MinVersion` to `tls.VersionTLS13`.

> **Note**
>
> Some AWS services do not yet support TLS 1.3. Setting TLS 1.3 as your minimum version might affect interoperability. We recommend testing this change with each service before deploying to production.

Document history for the AWS SDK for Go v2 Developer Guide

The following table describes the documentation releases for AWS SDK for Go v2.

Change	Description	Date
HTTP Interceptors	New section covering HTTP interceptors, their advantages over middleware, available hooks, and registration methods.	September 4, 2025
Data integrity protection with checksums	Content updated with details about automatic checksum calculation.	January 16, 2025
Initial release	Initial release of the AWS SDK for Go v2 Developer Guide	October 31, 2024